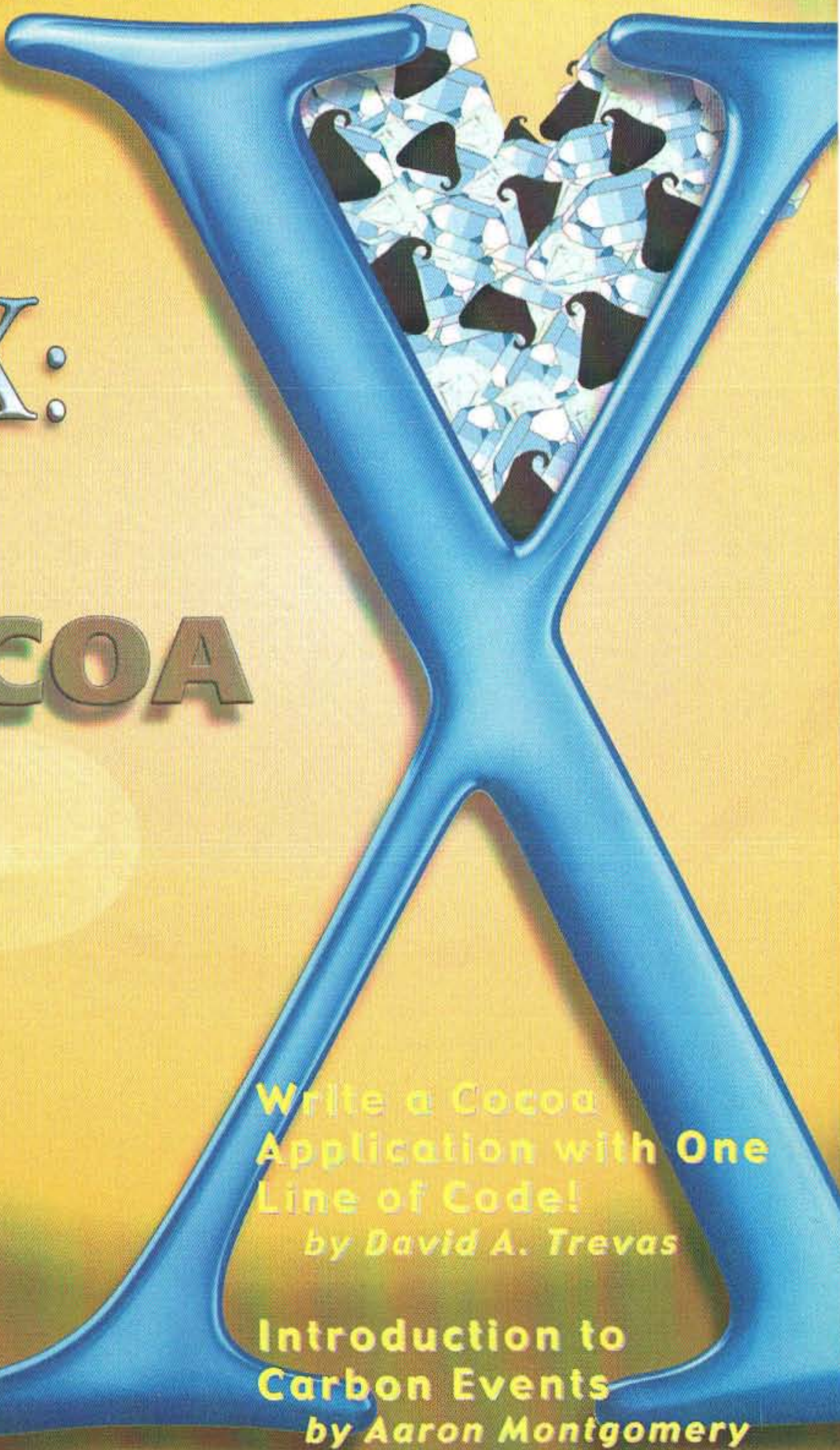
MacTech Magazine
Vol. 17, No. 07 • July 2001

MacTech®

The Journal of Macintosh Technology and Development



Mac OS X: CARBON & COCOA

Write a Cocoa Application with One Line of Code!
by David A. Trevas

Introduction to Carbon Events
by Aaron Montgomery

8.95 US
12.95 Canada
SSN 1067-8360
rinted in U.S.A.

LA to New York

(and Seattle and Miami)

in 150 milliseconds or less*

Edit View Special

Your Application ⌘N

Multi-point Meetings

Real-time Collaboration

Virtual Whiteboard™

Instant Assessment™

Best-quality video

Multimedia Presentation

Call for a TeraMedia® demo

TeraGlobal® COMMUNICATIONS

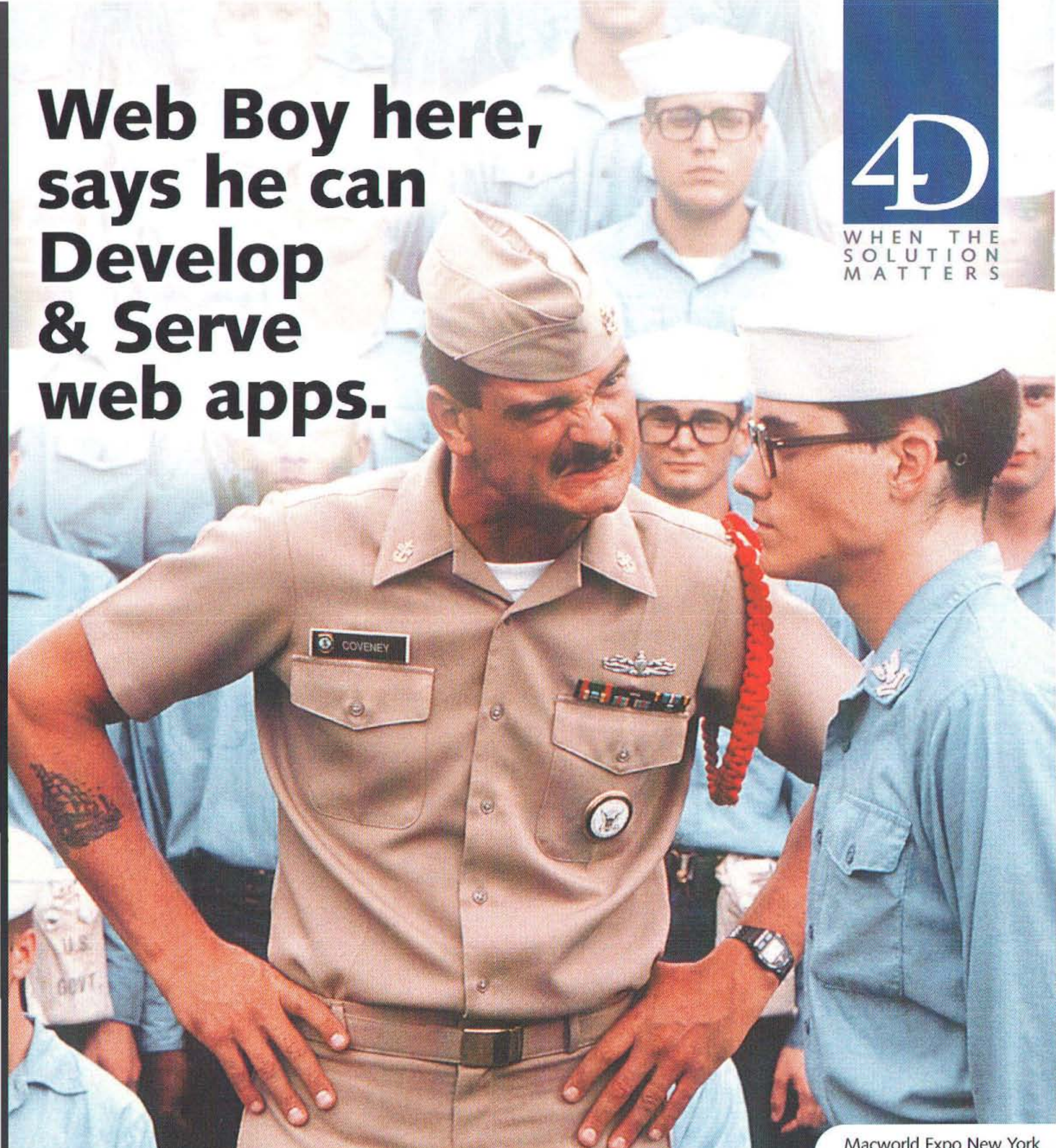
(858) 404.5500 ext 5660
www.teraglobal.com
sales@teraglobal.com

Digital DNA from Motorola

Interact. Collaborate. It's highest-quality multi-point video conferencing **and** true real-time multimedia collaboration. It's all done in software. Any number of users. Over any distance. At any time. In real time. It flies, you don't.

TeraMedia® by TeraGlobal

* Actual video latency varies with bandwidth. In most cases, TeraMedia® latency is below the threshold of human perception.


Web Boy here, says he can Develop & Serve web apps.
4D
WHEN THE SOLUTION MATTERS
COVENEY

Macworld Expo New York
BOOTH 458

INTRODUCING

4D 6.7 Web Edition $449

BUNDLED FOR A LIMITED TIME WITH ADOBE® GOLIVE® 5.0

Professional web application development from design to finish.

1.800.881.3466
www.4D.com/webedition

MacTech®

The Journal of Macintosh Technology & Development

A publication of **XPLAIN** CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin
Managing Editor • Jessica Stubblefield
Online Editor • Jeff Clites

Regular Columnists

Getting Started – Networking by John C. Welch
Programmer's Challenge by Bob Boonstra
MacTech Online by Jeff Clites
From the Factory Floor by Metrowerks

Regular Contributors

Vicki Brown, Andrew Stone, Tim Monroe, Erick Tejkowski, Kas Thomas, Will Porter, Paul E. Sevinç, Tom Djajadiningrat, and Jordan Dea-Mattson

MacTech's Board of Advisors

Dave Mark, Jordan Dea-Mattson, Jim Straus, Jon Wiederspan, and Eric Gundrum

PRINTED WITH SOYINK

This publication is printed on paper with recycled content.

MacTech's Contributing Editors

- Jim Black
- Michael Brian Bentley
- Tantek Çelik, Microsoft Corporation
- Marshall Clow
- John. C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Cobalt Networks
- John Hanay, Apple Computer
- Lorca Hanns, San Francisco State University
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Scott Knaster, Microsoft
- Mark Kriegsman, Clearway Technologies
- Peter N. Lewis, Stairways Software
- Bill McGlasson, Apple Computer
- Rich Morin
- Terje Norderhaug, Media*Design-In-Progress
- Nathan Nunn, Purity Software
- John O'Fallon, Maxum Development
- Alan Oppenheimer, Open Door Networks
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Dori Smith
- Andrew C. Stone, www.stone.com
- Michael Swan, Neon Software
- Chuck Von Rospach, Plaidworks
- Bill von Hagen
- Eric Zelenka

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin
President • Andrea J. Sniderman
Controller • Michael Friedman
Production Manager • Jessica Stubblefield
Production/Layout • W2 Graphics
Marketing Managers
Nick DeMello and Alyse Yourist
Events Manager • Susan M. Worley
Network Administrator • Ben Baumer
Accounting • Jan Webber, Marcie Moriarty
Customer Relations • Laura Lane
Susan Pomrantz
Shipping/Receiving • Joel Licardie

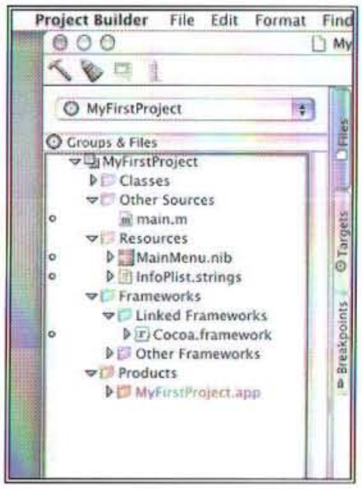
Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

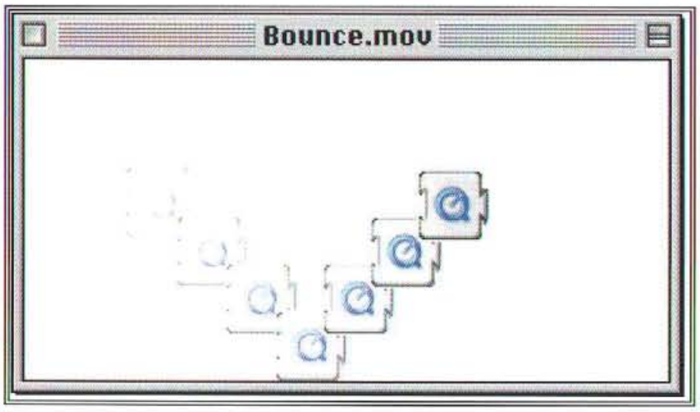
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

July 2001 • Volume 17, Issue 07



Write a Cocoa App . . . . Page 22



Back in Action...Page 54

By Andrew S. Downs

Is it Over Already?

What just happened?

MacHack 2001 (also known as MacHack 16) is now just a backed-up date book entry in my Palm VII, but strong memories remain. This was the most successful MacHack ever in several categories: a record number of attendees, a record number of papers presented, the first-ever Lego Mindstorms sessions track, a maximum number of student registrations, the longest Hack Contest ever (six hours), the largest keynote address (a panel of seven), and the first-ever Fireside Chat with Steve Wozniak.

What is this MacHack Thing?

If you have never heard of MacHack, let me try to explain: it is the Macintosh developer's conference with "underground cred" (per ZDNet eWEEK: www.zdnet.com/eweek), three-plus days of mostly technical sessions presented by the people who write the software you use every day, capped-off by a rousing display of showmanship and programming ability known as the Hack Contest, which is sponsored by The MacHax Group (www.machax.com).

Due to the round-the-clock programming effort and accompanying lack of sleep, many attendees take a while to decompress from the experience. Although I slept more this year than previously, it still took several days of thinking about what I'd seen (and done) to put things in perspective. As usual, I did not attend as many sessions as I would have liked, but I enjoyed everything I did attend.

Record-Setting Keynote Discussion

The conference traditionally opens with a keynote address at midnight of the first day. The keynote this year consisted of a panel of seven members of the original Macintosh development team: Bill Atkinson, Andy Hertzfeld, Jef Raskin, Caroline Rose, Donn Denman, Dan Kottke, and Randy Wigginton. Programming legend and author Scott Knaster worked the microphone as moderator, getting the discussion started and keeping things moving. Each of the panel members first told the tale of how he or she had come to be on the Macintosh team. Even if you have read various accounts elsewhere, there is something authentic about hearing the stories firsthand.

I am sorry to say I didn't make it through the whole keynote this year. I saw Jef Raskin in the elevator the next morning and he said he hadn't made it back to his room until after 6am!

Lego my, um, Lego

This year I made sure to get a peek at some of the Yoot sessions, specifically the ones focused on Lego Mindstorms. The sessions I attended were presented by Dave Baum. (There were adult Mindstorms sessions too. If you are unfamiliar with the Mindstorms concept, take a look at Matthew Nathan's article in the May 2001 issue of MacTech.) I was impressed with the ability of these young hackers, and I'm sure some of them can program circles around me. The Mindstorms sessions were held in one of the conference rooms, with four sets of robot parts to hack with and four iMacs for writing the control programs.

After a brief overview of the robot hardware and the programming environment (using a C-like language and compiler), the Yoots were walked through some simple programs that demonstrated both how to make the robot move in various directions for a certain length of time, and also how to use the optical sensor to determine proximity. Then, the hard part: the instructor tasked the kids with writing a program that enabled the robot to follow a curved track, stopping upon reaching the end. The track was a strip of gray tape laid upon white poster board. This was no easy feat, but I think all of the kids in that session managed to make it work. And yes, I did help out a little. What can I say, both the Yoots and the robots are pretty fun!

I attended another of Dave's sessions, this one geared toward adults. We had the same task as the Yoots: write a program to make the robot follow a line on the ground. In the true spirit of MacHack, Dave made it a competition: the robots would be timed during their traversal of the course, and the fastest finisher would be declared the winner. Obviously there was more to this than "go forward for three... 7 seconds". I had the good sense to team up with Jesse Donaldson from Palm Computing, and we spent some time improving upon a simple algorithm for moving the robot until it sensed a change in the track via the optical sensor. Upon sensing a change, the robot had to continue to follow the track, but with only one sensor this involved a fair amount of trial and error. Our entry eventually took second place after Jesse did some additional optimization; Dr. Waldemar Horwat finished first.

You can expect to see Mindstorms at MacHack again next year, maybe with a twist.

Andrew Downs is a four-time MacHack attendee, putting him somewhere above "babe-in-arms" and below "grizzled veteran". Let's just say he really enjoys the conference. Andrew works for Snippets Software, writing Internet apps for the desktop and PDAs. Andrew also teaches programming courses at Tulane University College in New Orleans, LA. You can reach him at andrew@downs.ws.

Emulation

I attended two sessions on emulation this year. Eric Traut from Connectix (www.connectix.com) spoke about Virtual PC, its evolution and some of the cool things being done in the latest release. Eric also discussed Connectix' newest product, Virtual PC for Windows. This is one of the most well-attended sessions at each conference. Eric is a terrific speaker, and casually and politely talks tech-stuff with the audience rather than at them. These sessions always inspire me.

Darek Mihocka from Emulators Inc. (www.emulators.com) also presented a session on emulation, but from a different angle: running Mac OS on Windows. Darek discussed some of the issues involved with emulating both the 68K and PowerPC instruction sets using the x86 architecture (quite a challenge!) After accuracy, the biggest issue is speed, and Darek provided some insight into how an emulator can run as fast as the real thing. Ouch!

Other Fun Stuff

A couple of other sessions that I enjoyed included Dave Koziol's Palm development overview, and Matt Morse's HeaderDoc discussion. Dave did a good job of single-handedly carrying the Palm track this year, assisted to some degree by the two Palm papers presented at the conference. Two features that make Palm development fun are its emphasis on small programs, and the fact that it is very Mac-like (which is no accident, given its pedigree).

HeaderDoc is Apple's open source header file documentation generator. I arrived late for this session, but had a good discussion with Matt afterward. That is another hallmark of the conference: you can have one-on-one conversations with engineers from Apple (and other companies) about the tools you use everyday. I contributed some bug fixes to HeaderDoc earlier this year, and this was the first time I physically met anyone else involved on the project.

And the Award for Best Paper Goes To...

Papers are one of the cornerstones upon which the conference is built. This year thirteen papers were presented, covering a wide variety of topics including information reliability, Palm development, data alignment, color image conversion, data sharing, OS X, scripting Cocoa, frameworks, and REALbasic development. The paper voted Best of Show was Ian Ollman's "Practical Altivec Strategies". In return Ian gets to attend next year's MacHack without paying the conference registration fee. Thanks Ian!

The Hack Contest

The sessions are a prelude to the hack contest on Friday night, starting at midnight. Here all of the hard work of the previous year (or several hours, if you get a late start) comes together. This year witnessed a record number of hacks presented at the contest, and though not all of them worked well (if at all), it reinforced the cutting-edge spirit of the conference. Over the past few years a growing number of Yoot hacks have been submitted, and this year I noticed what seemed to be a larger number of collaborative hacks. In fact, the winning hack (Apple Turnover) was a collaboration between Mac Murrett and Allon Stern. Apple Turnover rotates the image on your main display in real-time. Needless to say, editing a document then becomes very challenging!

Many of the hacks are Mac OS user interface related, since that allows for a high-level of showmanship, which is an important part of your presentation at 5am. Both Classic and OS X are represented. There are also a variety of videos, scripts, and Palm OS hacks; this year's contest also saw the first Mindstorms hacks. The hacks are available on the conference CD; point your browser to www.machack.com for more info.

But, I Missed It!

If you were unable to attend this year, cheer up. Like many good things, MacHack will come again. In the meantime, buoy your spirits by visiting the MacHack website (www.machack.com) and picking up a copy of the conference CD (which contains the hacks, sessions, and papers), browsing the press releases and conference information from this year, and dreaming about how much fun it will be in 2002. And start thinking about an entry for the Hack Contest. See you there! MT

AreaList Pro 7.6

Professional List Management for 4th Dimension

- **Display and edit arrays or fields**
- **Drag cells, rows or columns**
- **Gives complete control of color, font, size and style**
- **User resizable columns**
- **Mix field display with calculated columns**
- **Sorting on related-one fields**
- **Full navigation key support**
- **Quick setup with Advanced Properties dialog**

AreaList Pro is a plug-in package for 4th Dimension that provides editable, multi-column scrolling lists directly on your 4D application form. It equips you with the necessary tools to develop lists quickly and with increased flexibility. All features can be developed programmatically with AreaList Pro's rich command syntax, or through the Advanced Properties Dialog for easy point and click setup.

ASG AUTOMATED SOLUTIONS GROUP

Automated Solutions Group
Phone (714) 375-4252 • Fax (714) 848-0382
E-mail: sales@asgsoft.com • www.asgsoft.com
Toll-free (800) 375-4ASG

By John C. Welch

WWDC 2001

A peek inside the Apple

Welcome to the Show

The 2001 Apple World Wide Developer Conference is finished, and for those of us who were there, it was quite the whirlwind of activity and information. Like any other WWDC, this is the best chance for developers and technical people on the Mac to get a look inside the OS and the hardware, and to talk to the folks who make both. What was different about this session was the focus. No longer were the sessions about things that were about to ship, things that hadn't happened yet. No great charts about proposed shipping schedules. This was about the present, not the future, and it was almost all about Mac OS X.

When I say almost, I mean that discounting feedback forums, there were perhaps a handful of sessions that didn't explicitly talk about Mac OS X. It was a refreshing change of pace too. No 'maybe', 'will', or 'may'. Just 'are' and 'has'. So, from that point of view, there were no earth shattering surprises. Which is just as well, as the last thing a programmer, or an IS manager wants to hear is 'surprise'. That's not to say the conference was dull as dirt, but that you could pretty much anticipate what you were going to be seeing. My only real regret was that I had to miss all of Friday's sessions, and that I couldn't temporarily duplicate myself.

The Keynote

As all of you have seen and heard by now, the WWDC 2001 keynote was less of a revelatory one, and more of a lecture. Steve said it point blank, if you don't have your applications running native in Mac OS X, then your customers will find applications that are. This was not exactly taken well by some, but then again, harsh news rarely is. Now, some say Steve may have gone overboard with the finger shaking on this point, but maybe not. There are some folks, namely Adobe that have not only barely released a Carbon product, (Acrobat Reader is the best they can do?), but have been completely silent on even a vague schedule for when we can expect anything else. I understand about things like Acrobat and Photoshop that need scanner support, but things like GoLive and Illustrator should be almost done, or if Macromedia is any example, at least one of them should be done by now. But the point was made. The time is now to get product out there into the hands of the folks who write the checks, or someone else will come along and take your money. Apple understands painfully well how fickle loyalty can be in the computer market.

Not all was lecture and doom and gloom though. Jobs announced that the next version of Mac OS X Server was shipping, now sporting the Aqua interface, and taking its place as the server version of Apple's OS family. This version is also the product that marries the best of both the previous Mac OS X Server, and AppleShareIP. The other announcement was that as of May 21st, all Apple computers will ship with both Mac OS X, and Mac OS 9.1 installed, although Mac OS 9.1 will be the default boot OS.

This decision has been greeted by a curious range of emotions, ranging from those who think it is a complete mistake, to those who thing that it's an excellent idea. For myself, I'm somewhere in the middle. I think that Mac OS X still needs a lot of work in areas that haven't received as much press as the popular, yet somewhat inconsequential CD - burning/DVD Playback. Things like the fact that if you put your PowerBook to sleep without an external monitor or keyboard, when you wake it up, it doesn't recognize those things until you reboot, or that PC Card support is still ridiculously spotty. The lack of AppleScript support in the OS seems to not get a notice either.

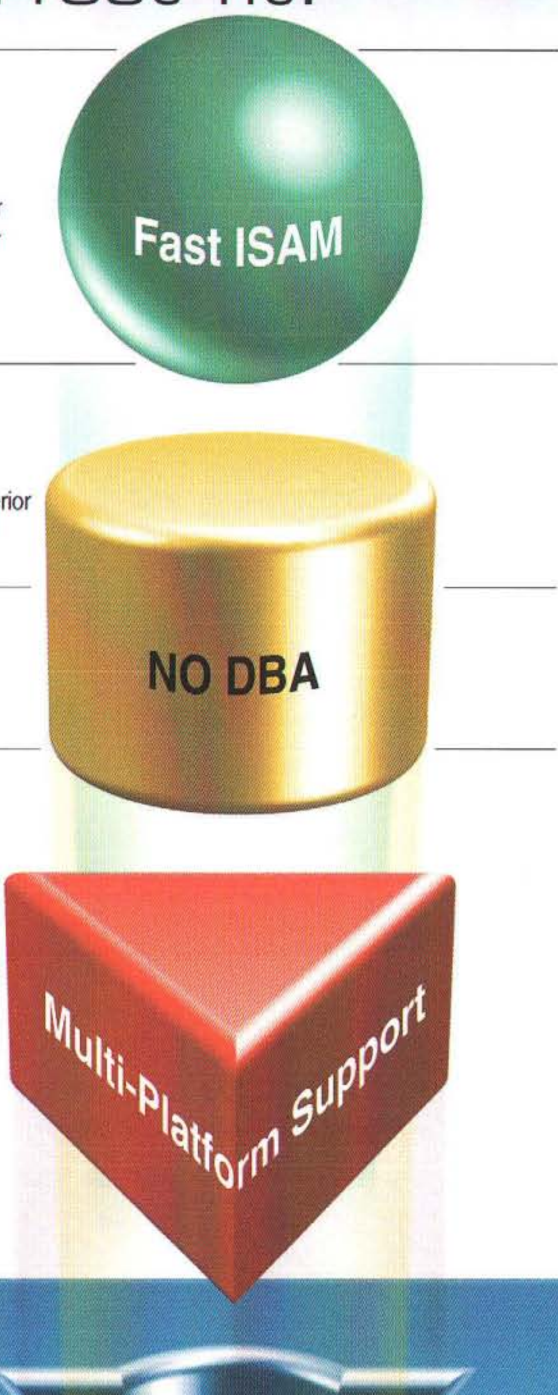
Still, I think that Apple needed to do this. It allows Mac OS X to truly be a shipping OS, and it says that Apple has enough confidence in the OS as it stands to really put it out in the world. By making the default boot OS Mac OS 9.1, it also allows the period between the WWDC and MacWorld Expo New York, to be a transitional one, giving folks time to get used to the fact that Mac OS X really is the future, and that it really is going to be the shipping OS for Apple. If Apple can get Mac OS X updated enough by July for MacWorld, then

John Welch <jwelch@completemac.com> is Training Specialist for Complete Mac Seminars, the leading MacOS instructional company. He has over fifteen years of experience at making computers work. His specialties are figuring out ways to make the Mac do what nobody thinks it can, and helping other folks learn how to do those things for themselves.

Looking for more speed, scalability and reliability from your database? c-tree Plus® is the perfect fit!

What shape is your database project? A tiny embedded database that requires a small footprint but demands rigorous functionality? Or a huge multi-platform, multi-user, multi-headache project? Have you struggled to find a better solution for your data access requirements, only to feel like you're sacrificing one requirement over another?

FairCom has delivered uncompromising database technology to commercial developers for over twenty years, with the primary goal of keeping control in the hands of the developer. Our customers rely on the speed, flexibility, scalability and reliability of c-tree Plus. The high performance and low cost of ownership make c-tree Plus an excellent choice for all sizes of database development projects.



Proven Database Technology

Besides the ISAM-level control and speed, perhaps the most important reason why small development houses and Fortune 1000 companies have chosen FairCom technology is the superior service offered by our sales and support staff. Our development staff understands the challenges you face, and we'll put our twenty years of experience to work to help you build better solutions.

Single Solution for Diverse Implementation

- Vertical Markets
- Embedded Systems
- Web & ASP Markets
- E-Commerce
- Smart-Cards
- Web-Enabled Appliances

Comprehensive Feature Set

- Royalty-free single user and multi-user support with full source code
- Client/server and custom server support
- Robust server side SDK - build your own application specific database server with full source available!
- Thread compliant + portable thread API
- Full featured transaction processing with savepoints, abort and full rollback
- Comprehensive security and encryption features
- Small footprint
- Fixed and variable length records and keys
- Store any data type - up to 18 million terabytes!
- Dynamic space reclamation
- ODBC and Crystal Reports™ drivers
- Full ISAM functionality
- Powerful stand-alone index support

Multiple-Platform Support

Mac OS, Mac OSX, MkLinux, Linux (PPC,Intel, SPARC, Alpha), Windows 95/98/ME/2000/NT, Novell Netware, Solaris (SPARC, Intel), Sun OS, OS/2, AIX, HP UX, SCO UnixWare, Interactive, AT&T Sys V, QNX, 88OPEN, FreeBSD, Lynx, Banyan Vines, and more...

Supports ADSP, SPX, TCP/IP

FairCom®
QUALITY•THROUGH•EXPERIENCE

www.faircom.com/maca

FairCom Offices

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

Mac Support Since 1985 • USA. 800.234.8180 • info@faircom.com

Fetch 4.0

Off the Leash

X

Fetchsoftworks.com

New Version • New Company

Same Author

making it the default OS for all Macs shipping after that event will not be as traumatic, or new.

A third keynote announcement was that WebObjects 5, now 100% Java was shipping. While of more interest to commerce, and large database web site developers, it still has implications for all technical users, developer or other. This is Apple saying that Java is not only a peer language with Objective C, but that it is the *only* language for Apple's biggest application server. It is, for Apple, a commitment to Java as a language and a platform for getting work done, that in its own way, rivals the Java commitment of such companies as IBM. It's also an example of Apple 'eating its own dog food'. They have been saying since the first introduction of Mac OS X, that the OS will be one of, if not the premiere Java platform. By making WebObjects a Java - only environment, they are betting a huge part of Mac OS X on their ability to pull that off. Having played with some various Java applications that either ran terribly, or not at all in Mac OS 9.X, I don't think it's going to be hard to pull off.

The final announcement was that Apple would no longer be shipping CRT monitors. This was not a real surprise, although the 17" LCD display that Steve introduced was greeted with great enthusiasm. Apple has been heading towards LCDs faster than any other computer company, and by elminating all external CRT monitors, (leaving the iMac as the sole CRT product in the lineup), Apple has managed to simplify their inventory a bit. This also will save them some overhead on shipping, as a 21" CRT is neither light, nor cheap to ship.

There were of course, some product demos, such as Adobe Premiere, showing off some of its new features, and how making a movie under Mac OS X means you don't have to kill virtual memory, or networking. When it gets around to shipping, it should be a neat product. Micromat's Drive 10 utility was demonstrated, showing, for the most part, that even a utility can have an elegant, functional interface. There was a really nice demonstration of the native release of Macromedia's Freehand 10. The demo was run by Macromedia's Vice President of Marketing, Tom Hale. He announced that Mac OS X had allowed them to get performance improvements of up to 50% in some operations. The demo also made Job's point about getting the Mac OS X native versions out there, especially to Adobe. Unlike Photoshop, which is still the dominant player in it's space, Illustrator and Freehand have always been close competitors, with Illustrator's main advantage being the close integration it has with Adobe's other products. But between Freehand 10, and the upcoming native version of Canvas 8, Adobe could see Illustrator's market share drop if they can't have it shipping by July. Dominique Goupil, of FileMaker Inc. also ran through a dog and pony show of the newly carbonized FileMaker Pro 5.5. Although a database will get more than most application types from the plumbing in Mac OS X, they added a very neat feature, namely new support for PDF files. By integrating the native PDF support in Mac OS X

- Custom Destinations
- Electronic Transaction Processing
- Easy Trialware Creation
- Install or UnInstall

Get It Together With InstallerMaker™

- Installation From FTP or HTTP Sites

When Time Is Money, Get It Done Fast!

Build Smaller Installers – With StuffIt InstallerMaker, you'll build installers faster than ever before! Compress your installers an average of 15% smaller using the power of the StuffIt Engine®. Smaller installers download faster, provide added space on CDs and servers, and increase network bandwidth.

Quickly Create Demoware - Easily turn your application into a polished demo. Just set the number of days for the demo to be active, paste in the graphics, and you're done!

Archive Freshening - Automatically update your installer project file; eliminate repeated searches for modified files.

- Resource Installation
- Built-In Resource Compression

Scripting Saves Time - InstallerMaker provides full scriptability for all features. Automating the build process saves time and helps ensure the integrity of your installer.

- ShrinkWrap Disk Image Support
- Marketing Opportunities To Help You Make Money

Trialware in Minutes - Creating trialware is a breeze with InstallerMaker. With just a few clicks, and no extra coding, you can create trialware that is e-commerce ready.

Update in a Flash - Easily build intelligent "diff" files in installers to create small updaters for quick online distribution. From one simple installer, you can update 68k, PowerPC or FAT applications.

- Hierarchical Package Support
- Built-In Updaters

More Details Online - There's a lot more InstallerMaker can do for you. Get all the info at www.aladdinsys.com.

Aladdin Systems

with the QuickTime capabilities of FileMaker Pro, you can now 'play' PDF files in a FileMaker database as though they were QuickTime movies. Considering that PDF is now the de facto standard for document exchange and archiving, this ability gives FileMaker quite a feather in its bonnet. The final demo was of Tony Hawk's skateboarding game. It's a neat game, but still seems to resemble Quake with a skateboard. The only thought this left me with is that ever since the rise of Quake, Tomb Raider, and Myth, most computer games seem to be variants on those basic archetypes. Hopefully, this will change soon.

The Meat of the Week

WWDC is hard to talk about sometimes because there is just so much information to handle. Even accepting that I just can't be in all the sessions I want to be in, there is still enough to make one's brain hurt.

Almost all the sessions were packed almost to capacity. No one, it seems, can get enough information on Mac OS X, and Apple was dishing it out in quantity. Some sessions had enough information, that the Q&A periods were held out in the hallways. The only session I was at that wasn't jammed full was the Tech Docs Feedback forum. This was particularly amusing to me, as one of the most consistent complaints about Apple is in this very area, yet the attendance doesn't reflect this. In any case, this was, as were the rest of the sessions, well worth the time taken. The tech docs folks not only were able to show where they had made progress on issues raised in the past, but also had a clear roadmap of where they were going. They said all the right things, but not just pro forma. There was a definite ring of sincerity to what they were saying, and the fact that they could point to definite improvements helped a lot here as well. Many issues surrounding non-developer technical documentation are being handled in what looks like the correct manner, that is, by handing that area over to the iServices folks, who are also in charge of certification and technical training on Mac OS X and Mac OS X Server. Although it may seem odd to praise a 'passing of the buck' as it were, the fact is, by ensuring that the proper groups are handling the different documentation areas, we, as the users of this documentation, get a better product that is more focused on our needs.

Another perennial capacity forum is the 'Meet the VPs' feedback forum. This is where attendees get a chance to be in the same room as the folks who head up the major groups at Apple, and talk to them directly. Comparing this year to sessions in past, the responsiveness is getting better each year, and the folks who came in from NeXT seem to be 'getting' the Mac more each year. Concerns about AppleScript were acknowledged, and for the first time, the NeXT folks were just as emphatic that AppleScript will be receiving the attention it needs to be the same tool as it is in Mac OS 9. This is important to many developers and technical people, as there have been, and still are fears that the NeXT folks don't 'get' the Mac, and don't want to get the Mac. By having those people take the lead in correcting this, a lot of fears can be calmed down. There are feedback forums at the WWDC for almost every area of the Mac OS, and they are almost all well attended. As one of the forum panel folks told me after a particularly emotional session, Apple values the good and the bad feedback, especially the passionate feedback. Developers only get passionate about an OS they care about, not one they are about to dump. So Apple realizes that they cannot afford to ignore, or not listen to a developer that gets a little carried away in the heat of the moment.

The technical sessions I attended were all well done, and well thought out. The information was presented in an interesting fashion, and considering how dry API explanations can be, the fact that these folks make it interesting shows the amount of care put into the WWDC by Apple. One of the amusing things for me was watching the student and open source developers realize that when Apple says it doesn't discuss unannounced products, it isn't kidding. Although this may seem to be an oxymoron for a developers conference, there's some sense there. Things are always changing, patches are always being released, etc. If Apple spends large amounts of time on what may happen in the future, they aren't spending time talking about what developers need to know for what is shipping *now*. While finding out tidbits about new sparklies on the horizon is cool, given the choice between talking about soon and now, I'll take now.

This focus was a good thing, especially considering the sheer volume of information presented. Many of the sessions included excellent advice on optimizing application performance, specific code examples, etc. Even better, since all of this information was about shipping products, it was all stuff that we can use here and now, not at some unspecified point in the future, and it might change. When Apple said, this is the API to use for this functionality, and here's where all the details are, it was nice to know that it wasn't all going to change overnight. Which is what we have all needed on Mac OS X for a while now. Reliable, accurate, detailed information. Is Apple 100% of where they need to be for now? No, but they are much closer than they were last year. Progress always beats stagnation, especially when talking about documentation.

Conclusion

I know that I'm not going into excruciating detail about session content, etc. But then, all of what was said at the conference is available from Apple online, or via email. What is different, and what cannot be replicated in any article is the value one gets from bouncing ideas off of Apple engineers, and other developers. You have to be there for that. If you are in a position where you need technical information on the Mac and the Mac OS, then you need to be at the WWDC. It's a lot of work, and a lot of fun. MT

Tired of Quicken®?

Appgen software runs on
Mac OS X
Linux and Windows

You now have the power to choose from this proven family of true accounting products. Visit www.appgen.com or call 1 800 231 0062 for more information about our unbeatable CPA Referral Program

APPGEN® Moneydance®

"As a user, Moneydance picks up where Quicken leaves off; as a CPA, Moneydance is an easier, more reliable, and safer way of maintaining your data."
—Peter J. Renzulli, CPA
Member AICPA

APPGEN® MyBooks®

True business accounting with audit trails for Small Office, Home Office and other small businesses with 1-5 users. And it's modifiable!

APPGEN® Accounting for Small Business

Fully customizable business accounting software for small businesses with up to hundreds of users

APPGEN® PowerWindows™ v5.1

The four-star rated software that forms the basis for all Appgen business products

from The Appgen Software Companies

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

DOWN-N-OUT

George Warner earns two Challenge points for suggesting another interesting board game, this time the solitaire game known as Down-N-Out. The Down-N-Out board is a 10x30 rectangular array of cells, initially populated randomly with 100 cells of each of three colors. The object is to score as many points as possible by removing cells from the board. A cell can be removed if it is adjacent (horizontally or vertically, but not diagonally) with a cell of the same color. When a cell is removed, all cells connected to it by transitive adjacency are removed. That is, all adjacent cells are removed, and all cells adjacent to those cells, etc. The number of points earned for each move is equal to the square of the number of cells removed (e.g., 2 cells = 4 points, 3 cells = 9 points, etc.). There is an obvious advantage to planning moves that maximize the number of connected cells removed simultaneously. After each move, the board is compacted by sliding all cells downward to fill any empty cells, and then by sliding all columns to the center to fill any empty columns. The game continues as long as cells can be removed.

For those of you interested in trying the game, there is a shareware version available at <http://www.peciva.com/software/downout.shtml>.

The prototype for the code you should write is:

```
typedef char CellColor;   /* 0==empty, 1..numColors are valid colors */

void InitDownNOut(
   short boardSizeRows, /* number of rows in the game */
   short boardSizeCols, /* number of columns in the game */
   short numColors,          /* number of colors in the game */
   WindowPtr wdw   /* window where results of your moves should be displayed */
);

void HandleUpdateEvent(EventRecord theEvent);

Boolean /* able to play    */ PlayOneDownNOutMove(
   CellColor board[], /* board[row*boardSizeCols + col] is color of cell at
[row][col] */
   long score,               /* points earned prior to this move */
   short *moveRow,           /* return row of your next move */
   short *moveCol            /* return col of your next move */
);

void TermDownNOut(void);
```

Each game begins with a call to your InitDownNOut routine, where you are given the dimensions of the game board (boardSizeRows and boardSizeCols), the number of colors in the game (numColors), and a pointer (gameWindow) to a WindowRecord where you must display the game state as it progresses. Finally, you will be given the initial state of the game board, fully populated with equal numbers of each color cell, subject to rounding limitations. InitDownNOut should allocate any dynamic memory needed by your solution, and that memory should be returned at the end of the game when your TermDownNOut routine is called.

Your PlayOneDownNOutMove routine will be called repeatedly, once for each move you make. You will be given your current point score as calculated by the test code and the state of the game board. You should determine the most advantageous move and return it in moveRow and moveCol. You should update the game board, eliminating cells removed by your move and compacting the board vertically and then horizontally. You should calculate the number of cells removed and return it in numberOfCellsRemoved.

The last time we ran a Challenge that involved maintaining a display, contestants asked how the window would be redrawn in response to an update event. This time, I'm asking you to write a routine to do that. Your HandleUpdateEvent routine will be called by the test code whenever an update event is received for your gameWindow.

During the call to InitDownNOut, and after each of your moves, you should display the updated game state in the gameWindow. The details of the display are up to you, as long as the display correctly and completely represents the state of the board.

The winner will be the best scoring entry, as determined by the sum of the point score of each game, minus a penalty of 1% for each millisecond of execution time used for that game. The Challenge prize will be divided between the overall winner and the best scoring entry from a contestant that has not won the Challenge recently.

This will be a native PowerPC Challenge, using the CodeWarrior Pro 6 environment. Solutions may be coded in C or C++. I've deleted Pascal from the list of permissible languages, both because it isn't supported by CW6 (without heroics) and because no one has submitted a Pascal solution in a long time.

THREE MONTHS AGO WINNER

Congratulations to **Ernst Munter** (Kanata, Ontario, Canada) for submitting the best scoring solution in the April **Crossword II** Challenge. This Challenge was inspired by a classroom exercise to construct a 20x20 crossword puzzle using the names of the elements in the periodic table, valuing each word according to the atomic number of the corresponding element, with the objective of maximizing the total value of the puzzle. We generalized the problem by making the word list, word values, and puzzle size parameters of the problem. And to incorporate the usual emphasis on efficiency, we penalized each test case by 1% for each minute of execution time required to generate the puzzle.

The winning solution starts by assigning a strength value to each word in the word list. The strength of a word is a scaled version of the value assigned by the problem input, divided by the length of a word. This heuristic favors shorter words of a given value over longer words of the same value.

Then the Board::Solve routine tries to place words until a time limit (set to 15 seconds) expires or there are no more valid moves to explore. The moves are attempted in order of decreasing value, where the value of a move is the assigned value of word being placed, divided by the length of the word minus the number of letters that intersect other words. Again, this gives priority to placement of shorter high value words over longer ones, and to placements that efficiently use the board space by intersecting other words.

I evaluated the four entries received using a set of ten test cases ranging in size from 20x20 to 50x50. Ernst's solution packed 10% more word value into his puzzles than the second place entry by **Ron Nepsund**, taking significantly more execution time as well. For the original 20x20 problem based on the periodic table, Ernst's entry produced the following crossword, valued at 2470 points:

```
L A W R E N C I U M     X E N O N   P
  S       E           B             L
  T       O           A M E R I C I U M
  A C T I N I U M     R             T
  T             E     I       A R G O N
S I L V E R   E R B I U M       H   N
  N             C     M         E   I
  E     B I S M U T H   P O L O N I U M
                R     G         I   M
  F   C         Y     O   R     U
  E   A   C           L E A D   M   U
  R A D I U M     T   D   D   B     R
  M   M   R       H       O   E     A
  I R I D I U M   O   T I N   R     N
  U   U   U       R           K     I
```

```
  M   M   M   I   I     N O B E L I U M
              R   U           L     M
C E R I U M   O S M I U M   Z I N C
              N               U
H A F N I U M   F R A N C I U M
```

Ernst would have won by an even wider margin were it not for an ambiguity in the problem statement. The problem specified that each word could only occur once in the puzzle, and that each sequence of letters in the puzzle had to form a word. What I meant to say, however, but didn't, was that each word in the puzzle needed to be distinct. Two of the contestants took advantage of this loophole, for example, to claim credit for the word "tin" embedded in the longer word "actinium". Fortunately for my sense of fairness if nothing else, when I ran the tests both allowing and not allowing the loophole, the scores were such that the ranking of the entries was unchanged. The results as presented reflect the actual wording of the puzzle, and allow a word to be embedded in another word.

As the best-placing entry from someone who has not won a Challenge in the past two years, **Ron Nepsund** wins a share of this month's Challenge prize. You don't need to defeat the Challenge points leaders to claim a part of the prize, so enter the Challenge and win Developer Depot credits!

The table below lists, for each of the solutions submitted, the number of points earned by each entry, and the total time in seconds. It also lists the code size, data size, and programming language used for each entry. As usual, the number in parentheses after the entrant's name is the total number of

Tri-platform Calendaring

OS X *Carbon*
Mac OS *7.1 - 9.x*
Windows *95 - 2000*

New!

CalendarMonster 1.2
Multi-platform calendar

- **End-user Calendar**
  The best FMP calendar experience
  - Fast, fast, fast!!!
  - Platform-appropriate user experience
  - Highly scalable
- **Excellent Developer Support**
  Quickly and easily integrate with your FileMaker Pro projects
  - 100% live data, 100% modular, no plug-ins
  - FileMaker Pro 4 or 5 (including 5.5)
  - As little as 2 ScriptMaker calls
  - Dedicated developers' utility

only $49! per user

Developer and volume licensing available

ASCENDING TECHNOLOGIES

monster@asctech.com
Download your own FREE fully-functional demo
http://www.asctech.com

Challenge points earned in all Challenges prior to this one.

Name	Points	Time (secs)	Code Size	Data Size	Lang
Ernst Munter (731)	52378	151.4	3940	174	C++
Ron Nepsund (47)	47489	6.7	44520	6530	C++
Jan Schotsman (7)	44888	32.9	12708	448	C++
Ken Slezak (26) [LATE]	4927937.4	3160	47	C++	

Top Contestants ...

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 20 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants, the number of wins over the past 24 months, and the total number of career Challenge points.

Rank	Name	Points (24 mo)	Wins (24 mo)	Total Points
1.	Munter, Ernst	304	12	751
2.	Rieken, Willeke	83	3	134
3.	Saxton, Tom	76	2	185
4.	Taylor, Jonathan	56	2	56
5.	Shearer, Rob	55	1	62
6.	Wihlborg, Claes	49	2	49
7.	Maurer, Sebastian	48	1	108

... and the Top Contestants Looking for a Recent Win

In order to give some recognition to other participants in the Challenge, we also list the high scores for contestants who have accumulated points without taking first place in a Challenge during the past two years. Listed here are all of those contestants who have accumulated 6 or more points during the past two years.

Rank	Name	Points (24 mo)	PointS Total
8.	Boring, Randy	32	142
9.	Schotsman, Jan	14	14
10.	Sadetsky, Gregory	12	14
11.	Nepsund, Ronald	10	57
12.	Day, Mark	10	30
13.	Jones, Dennis	10	22
14.	Downs, Andrew	10	12
15.	Duga, Brady	10	10
16.	Fazekas, Miklos	10	10
17.	Flowers, Sue	10	10
18.	Strout, Joe	10	10
19.	Nicolle, Ludovic	7	55
20.	Hala, Ladislav	7	7
21.	Miller, Mike	7	7
22.	Widyatama, Yudhi	7	7
23.	Heithcock, JG	6	43

get a head start

beta release

download now

Middleware for the Mac

Need a Better Software Communication Solution?

We Built the Missing Piece

RapidMQ

Messaging & Queuing

RapidMQ is a flexible Messaging and Queuing solution enabling applications to communicate securely and reliably. RapidMQ helps solve software communication problems in many different markets.

Business Systems

- Communicate with customers, vendors, and partners using industry standard SSL security.
- Control the message traffic within your network through the use of Connections and Zones.
- Collect and analyze message traffic using RapidAdmin software.
- Quickly integrate RapidMQ using easy to understand programmatic interfaces.

E-Commerce

- Real-time data synchronization allows data to be cached in memory for fast access.
- Messaging infrastructure connects web applications to internal systems.
- Higher return on investment (ROI) gives you a better value than competing middleware solutions.
- Customer data is safely transferred using industry standard SSL security.

Consumer Software

- Automatically collect registration information from software even when the computer was not connected to the Internet at the time of installation.
- Allow users to purchase new features or updates directly through your software using industry standard SSL security.
- Provide product feedback and bug reporting interfaces within your application for direct delivery to your internal systems.

Mac

Jiiva, Inc.
www.jiiva.com/rapidmq

Built for Mac OS X

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Ernst's winning CrosswordII solution:

CrosswordII.cp
Copyright © 2001
Ernst Munter, Kanata, ON, Canada

```
/*
Task
——
Make a crossword puzzle from a set of supplied words. The twist is: different
words have different point values, and the object is to maximize the point count.

Solution
————
The words are sorted according to "strength", that is the normalized point
count = points/length.

A search tree is built using the most promising placements, up to a maximum
of kMaxMoves, at each stage.

To avoid any significant point penalty (of 1% per minute), processing stops
after 15 seconds.

A private copy of the puzzle is built where each cell is an unsigned character,
with value of 0, c, or 2*c. An empty cell is 0, an placing a word is done
by adding each of the word's character into the corresponding cell. Similarly,
removal of a word is done with subtraction.

*/
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <Events.h>
#include "CrosswordII.h"

typedef unsigned long ulong;
typedef unsigned short ushort;
typedef unsigned char uchar;

static int N=0;

enum {
  kDown  = 0,
  kAcross  = 1,
  kMaxMoves = 5,
  kTicksPerSecond = 60,
  kMaxSeconds = 15
};
```

struct MyWord

```
struct MyWord
// Encapsulation of Words
{
  const Words* word;
  ulong  length;
  ulong  strength;// length-relative value
  bool used;
  MyWord(){}
  MyWord(const Words* wp) :
    word(wp),
    length(strlen(wp->theWord)),
    strength((0x10000L*wp->value)/(1+length)),
    used(false)
  {}
  const Words* Word() const {return word;}
  const char* Chars() const {return word->theWord;}
  long Value() const {return word->value;}
  int Length() const {return length;}
  bool IsAvailable() const {return !used;}
  void SetUsed() {used = true;}
  void ClearUsed() {used = false;}
  ulong Strength() const {return strength;}
};

static int CmpWord(const void* a,const void* b)
{
  MyWord* ap=(MyWord*)a;
  MyWord* bp=(MyWord*)b;
  return bp->strength - ap->strength;
}

struct MyMove
// A Move is a placement of a word
{
  MyWord* w;
  ulong  value;
  ushort row;
  ushort col;
  ushort delta;
  ushort size;
  ulong  Value() const {return value;}
  ulong  Points() const {return w->word->value;}
  void   Init(int numIntersects,MyWord* wx,
                         int r,int c,int d,int s)
  {
    w=wx;
    value=(0x10000 * w->word->value) /
                         (1+w->Length()-numIntersects);
    row=r;
    col=c;
    delta=d;
    size=s;
  }
  void   Clear() {value=0;}
  ulong  IsValid() const {return value;}// != 0
  void Convert(const Words* words,WordPositions* p)
  // Converts this instance of "MyMove" to a "WordPosition" as defined in
  // "CrosswordII.h"
  {
    p->whichWord=w->word-words;
    p->row=row;
    p->col=col;
    p->orientation=(delta==1)?kAcross:kDown;
  }
  void RemoveWord(char* puzzle)
  {
    char* p=puzzle+row*size+col;
    char* str=w->word->theWord;
    for (int i=0;i<w->Length();i++)
    {
      *p -= *str++;
      p+=delta;
    }
    w->ClearUsed();
  }
  void PlaceWord(char* puzzle)
  {
    char* p=puzzle+row*size+col;
    char* str=w->word->theWord;
    for (int i=0;i<w->Length();i++)
    {
      *p += *str++;
      p+=delta;
    }
    w->SetUsed();
  }
  int IntersectAcross(MyWord* w,int r,int c,
                char* puzzle,int puzzleSize)
  {
// returns -1(no fit), 0 (fit, no intersects) or n>0 (n intersects with other words)

    // insertion point p
```

Top 10 Reasons Why You Should Buy Funnel Web®

10. It's the most accurate Web site visitor analysis tool available

9. Everyone should own at least one product named after the world's deadliest spider

8. One of the developers once worked in the gear room of the world's highest aerial tram

7. Buy the product and we'll let you keep the CD

6. It produces reports so beautiful you'll want to carry them around in your wallet

5. It runs on Mac, Linux, Windows, FreeBSD and Solaris

4. Someone needed to start a new trend

3. Thousands of your colleagues are already using it

2. It's 400% faster than competing products*

And the **#1** reason why you should try Funnel Web Analyzer is...

(What, you think we're going to give it up on the first date? Go to www.quest.com/top10 to find out!)

*Refers to a recent independent review from Neiger Computer Consulting.

Funnel Web Analyzer is one part of our complete suite to map, manage and measure your Web site.

QUEST SOFTWARE

www.quest.com

United States 1.949.754.8000 • Germany +49.211.770967.0 • United Kingdom +44.1628.601000 • France +33.1.4131.96.96 • Australia +61.3.9811.8000

```
    char* p=puzzle+r*puzzleSize+c;
    int len=w->Length();

    // cell before the word must be a border or blank
    char* rowStart=puzzle+r*puzzleSize;
    char* cellBefore=p-1;
    if ((cellBefore >= rowStart) && (0 != *cellBefore))
      return -1;

    // cell after the word must be a border or blank
    char* rowEnd=rowStart+puzzleSize;
    char* cellAfter=p+len;
    if ((cellAfter < rowEnd) && (0 != *cellAfter))
      return -1;

    // all cells to the side of the word must be
    //    (a) either blank
    //    (b) or part of a crossing word
    // we know case b applies only if the cell the current word is
    // to occupy is already occupied - with a letter equal to str[x]

    char* str=w->word->theWord;
    char* puzzleEnd=puzzle+puzzleSize*puzzleSize;
    int numIntersects=0;
    for (int i=0;i<len;i++,str++,p++)
    {
      if (*p == 0)// crossing a blank
      {
        // cell above must be outside border, or blank
        char* cellAbove=p-puzzleSize;
        if ((cellAbove >= puzzle) && (0 != *cellAbove))
          return -1;
        // cell below must be outside border, or blank
        char* cellBelow=p+puzzleSize;
        if ((cellBelow < puzzleEnd) && (0 != *cellBelow))
          return -1;
      } else if (*p == *str)// crossing a word, matching
      {
        numIntersects++;
      } else // crossing, but no match
      {
        return -1;
      }
    }
    Init(numIntersects,w,r,c,1,puzzleSize);
    return numIntersects;
  }
  int IntersectDown(MyWord* w,int r,int c,
                char* puzzle,int puzzleSize)
  {
// returns -1(no fit), 0 (fit, no intersects) or n>0 (n intersects with other words)

    // insertion point p
    char* p=puzzle+r*puzzleSize+c;
    int len=w->Length();

    // cell before the word must be a border or blank
    char* colStart=puzzle+c;
    char* cellBefore=p-puzzleSize;
    if ((cellBefore >= colStart) && (0 != *cellBefore))
      return -1;

    // cell after the word must be a border or blank
    char* bottomBorder=puzzle+puzzleSize*puzzleSize;
    char* cellAfter=p+len*puzzleSize;
    if ((cellAfter < bottomBorder) && (0 != *cellAfter))
      return -1;

    // all cells to the side of the word must be
    //    (a) either blank
    //    (b) or part of a crossing word
    // we know case b applies only if the cell the current str would
    // occupy is already occupied - with a letter equal to str[x]

    char* str=w->word->theWord;
    char* puzzleEnd=puzzle+puzzleSize*puzzleSize;
    char* leftEdge=puzzle+r*puzzleSize;
    int numIntersects=0;
    for (int i=0; i<len;
            i++,str++,p+=puzzleSize,leftEdge+=puzzleSize)
    {
      if (*p == 0)// crossing a blank
      {
        // cell on the left must be the left border, or blank
        char* cellLeft=p-1;
        if ((cellLeft >= leftEdge) && (0 != *cellLeft))
          return -1;
        // cell on right must be on the right edge, or blank
        char* cellRight=p+1;
        char* rightEdge=leftEdge+puzzleSize;
        if ((cellRight < rightEdge) && (0 != *cellRight))
          return -1;
      } else if (*p == *str)// crossing a word, matching
      {
        numIntersects++;
      } else // crossing, but no match
      {
        return -1;
      }
    }
    Init(numIntersects,w,r,c,puzzleSize,puzzleSize);
    return numIntersects;
  }
};

typedef MyMove* MyMovePtr;

inline bool operator > (const MyMove & a,const MyMove & b)
{
  return a.Value() > b.Value();
}
```

struct MyMoveArray

```
struct MyMoveArray
{
  int numMoves;
  int maxMoves;
  MyMove moves[kMaxMoves];
  MyMoveArray(int max) :
    numMoves(0),
    maxMoves(max)
  {}
  int NumMoves() const {return numMoves;}
  MyMove* Moves() {return moves;}
  void Insert(MyMove & m)
  {
    if (numMoves==0)
    {
      numMoves=1;
      moves[0]=m;
    } else if (numMoves<maxMoves)
    {
      MyMove* mx=moves+numMoves;
      while ((mx>moves) && (*(mx-1) > m))
      {
        *mx=*(mx-1);
        mx=mx-1;
      }
      *mx=m;
      numMoves++;
    } else if (m > moves[numMoves-1])
    {
      numMoves--;
      Insert(m);
    }
  }
};
```

struct Board

```
struct Board
{
  long puzzleSize;
  char*  puzzle;
  long numWords;
  MyWord* myWords;
  long numPositions;
  WordPositions* bestPositions;

  MyMove*      movePool;// single pool allocated for movelists
  MyMove*      endMovePool;
  MyMovePtr*  moveStack;  // move stack tracks the history of executed moves
  MyMovePtr*  moveStackPointer;
  MyMovePtr*  lastMoveStack;
  Board(long pSize,const Words* words,long nWords) :
    puzzleSize(pSize),
    puzzle(new char[(pSize)*(pSize)]),
    numWords(nWords),
    myWords(new MyWord[nWords]),
    numPositions(0),
    bestPositions(new WordPositions[numWords]),


    movePool(new MyMove[numWords*kMaxMoves]),
    endMovePool(movePool+numWords*kMaxMoves),
    moveStack(new
MyMovePtr[numWords]),moveStackPointer(moveStack),
    lastMoveStack(moveStack+numWords-1)

  {
    for (long i=0;i<numWords;i++)
      myWords[i]=MyWord(words+i);

// sort words by strength
    qsort(myWords,numWords,sizeof(MyWord),CmpWord);

// remove all 0-value words
    long i=numWords;
    while ((i>0) && (myWords[i-1].Value()<=0))
      i=i-1;

    numWords=i;
  }
  ~Board()
  {
    delete [] bestPositions;
    delete [] myWords;
    delete [] puzzle;
  }
  void Clear()
  {
    memset(puzzle,0,sizeof(char)*(puzzleSize)*(puzzleSize));
  }
  int Solve(const Words* words,WordPositions* positions);

  void SetPosition(const Words* words,MyWord* w,WordPositions*
pos,
    int row,int col,int o)
  {
    pos->whichWord=w->Word()-words;
    pos->row=row;
    pos->col=col;
    pos->orientation=o;
  }

  void PushMove(MyMove* mp){
    *moveStackPointer++=mp;
  }

  MyMove* PopMove()
  {
    return *--moveStackPointer;
  }

  MyMove* GenerateMoveList(MyMove* mp)
  {
// Lists all legal moves in a list, starting with a null-move;
// sorts the moves and returns the highest value move on the list
// Each move is given a "value" reflecting its relative merit.
    if (mp+kMaxMoves >= endMovePool)
      return 0; // no room for movelist, should not really happen
                // but if it does, we just have to backtrack
    MyMove m;
    int i,row,col,maxRow,maxCol,drow,dcol;

// create moves
    MyMoveArray ma(kMaxMoves);

    MyWord* w=myWords;
    ulong bestStrength=0;
    for (i=0;i<numWords;i++,w++)
    {
      if (!w->IsAvailable()) continue;
      ulong strength=w->Strength();
      if (strength < bestStrength) continue;
      maxCol=maxRow=puzzleSize-w->Length();
```

http://www.scientific.com

Professional Software Developers

Looking for career opportunities?

Check out our website!

Nationwide Service

Employment Assistance

Resume Help

Marketability Assessment

Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)

das@scientific.com

Cross-Platform C++

PP2MFC puts your PowerPlant® applications on Windows®

www.oofile.com.au

```
// find every legal position
        drow=0;
        for
(row=puzzleSize/2;(row>=0)&&(row<puzzleSize);row+=drow)
        {
          dcol=0;
          for (col=puzzleSize/2;(col>=0) &&
(col<puzzleSize);col+=dcol)
          {
            if ((col<=maxCol) &&

(m.IntersectAcross(w,row,col,puzzle,puzzleSize)>=0))
            {
              ma.Insert(m);
              bestStrength=strength;
            }

            if ((row<=maxRow) &&

(m.IntersectDown(w,row,col,puzzle,puzzleSize)>=0))
            {
              ma.Insert(m);
              bestStrength=strength;
            }

            if (dcol>=0) dcol=-1-dcol; else dcol=1-dcol;
          }
          if (drow>=0) drow=-1-drow; else drow=1-drow;
        }
      }

// put a sentinel 0-move at the start of the movelist
      mp->Clear();
// copy moves from the moves array into the movelist space
      MyMove* mx=ma.Moves();
      for (int i=0;i<ma.NumMoves();i++)
        *(++mp) = *mx++;

      return mp;
    }

    long Execute(MyMove* mp)
    {
      mp->PlaceWord(puzzle);
      PushMove(mp);
      return mp->Points();
    }

    MyMove* Undo(long & points)
// Undoes the last stacked move, returns this move, or 0 if no move found
    {
      MyMove* mp=PopMove();
      if (mp==0) return mp;
      mp->RemoveWord(puzzle);
      points -= mp->Points();
      return mp;
    }

    long CopyMovesBack(const Words* words,WordPositions*
positions)
// Scans the movestack, converts MyMoves to positions.
// Returns the number of positions
    {
      int numMoves=0;
      for (MyMovePtr*
index=moveStack+1;index<moveStackPointer;index++)
      {
        MyMove* mp=*index;
        mp->Convert(words,positions+numMoves);
        numMoves++;
      }
      return numMoves;
    }
};
```

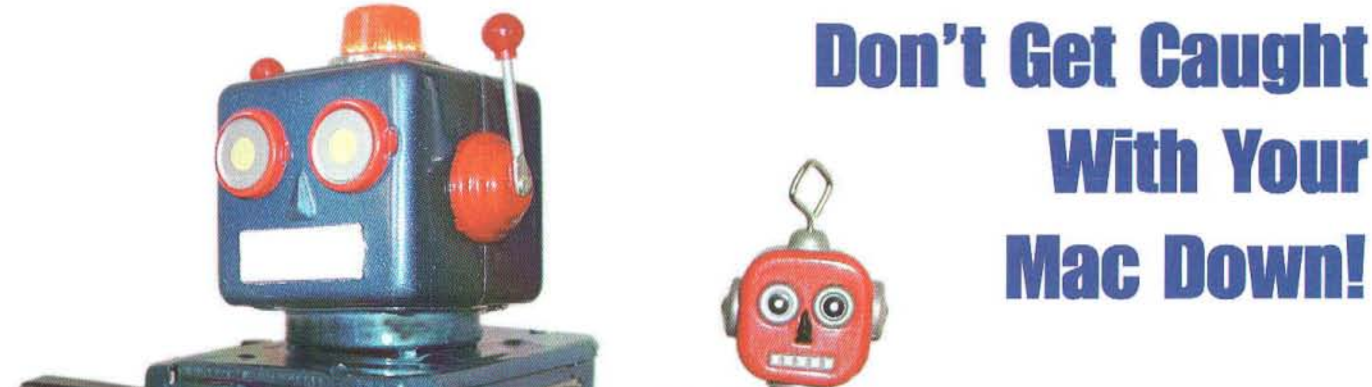
Board::Solve

```
int Board::Solve(const Words* words,WordPositions* positions)
{
  WordPositions* pos=positions;
  long numPositions=0;
  long bestPoints=0;
  long start=TickCount();

  Clear();
  moveStackPointer=moveStack;
  // Put a sentinel null move at start of move stack
  PushMove(0);
  MyMove* moveList=movePool;
  long points=0;

  MyMove* nextMove=GenerateMoveList(moveList);
  // moveList to nextMove defines a movelist which always starts with a 0-move
  // and is processed in order nextMove, nextMove-1, ... until 0-move is found
  if (!nextMove)
    return 0;

  for (;;)
  {
    while (nextMove && nextMove->IsValid())
    {
      points+=Execute(nextMove);
      if (points > bestPoints)
      {
        bestPoints=points;
        numPositions=CopyMovesBack(words,positions);
      }
      moveList=1+nextMove;
      long numTicks=TickCount()-start;
      if (numTicks>kMaxSeconds*kTicksPerSecond)
        break;
      nextMove=GenerateMoveList(moveList);

    } // end while

    do {
      MyMove* prevMove=Undo(points);
      if (!prevMove)    // stack is completely unwound, exhausted
        break;

    // try to use the last move:
      nextMove = prevMove-1;
    } while (!nextMove->IsValid());

    moveList=nextMove;
    while ((moveList>=movePool) && (moveList->IsValid()))
      moveList--;

    if (moveList<=movePool)
      break;
  }
  return numPositions;
}
```

CrosswordII

```
short /* numberOfWordPositions */ CrosswordII  (
  short puzzleSize,          /* puzzle has puzzleSize rows and columns */
  const Words words[],    /* words to be used to form the puzzle */
  short numWords,            /* number of words[] available */
  WordPositions positions[] /* placement of words in puzzle */
) {
  if (numWords <= 0)
    return 0;

  Board B(puzzleSize,words,numWords);

  long numberOfWordPositions=B.Solve(words,positions);

  return numberOfWordPositions;
}
```

A New System Has Arrived.



No Mac is complete without Timbuktu Pro, the premier remote control and file transfer software for Mac OS for over ten years.

No network is complete without the smart systems management of netOctopus.

Both tools are newly rewritten for OS X, bringing the power and simplicity of Netopia software to the world's most advanced operating system. For deployment, training and support of all your Macs (and PC's!) Timbuktu and netOctopus are indispensable.

Are you ready to migrate? We're ready to take you there.

macosxready.com

Timbuktu Pro®

netOctopus

By David A. Trevas

Write a Cocoa Application with One Line of Code!

With the advent of Mac OS X, there is whole new way to conceive and build Macintosh programs called Cocoa. While the Classic and Carbon environments provide a link to the single-digit Mac operating systems, Cocoa is for OS X only and, as such, takes maximum advantage of the technical and visual improvements of the new system. The use of the elegant Objective-C language makes the concepts of object-oriented programming very clear. No matter how elegant the language is, diving too quickly into writing many lines of code in an introductory program can discourage many of the newcomers to Cocoa. The reader should know how the program's author chose each line of code. It is this thought process that the readers need to learn. I believe that an introductory article should show you around and demonstrate how basic things work to give you the confidence and the knowledge to go to higher levels. To give you a good welcome to Cocoa, I have chosen an excellent number of lines of code for you to write: *one*. I promise that you're going to be surprised what you can do with just one line of code!

This article is intended to introduce you to the structure of a Cocoa project, the workflow involved in building one, and the techniques and terminology of Cocoa programming in general. I am assuming nothing about your knowledge of programming, but I am assuming that you have used some Macintosh applications and have used buttons, menus, text fields and the like. You will need perseverance–a trial-and-error attitude–since both of us will be very lucky if everything goes exactly right the first time, even if you follow the instructions carefully. You may have to try something a few times or even backtrack a little to get things right.

Create a New Project

If you don't already have ProjectBuilder, go to the Apple Developer Connection web site and join. "How much is this going to set me back?," you grumble, but the good news is that the lowest membership level that can access the Mac OS X Developer Tools is FREE! You can become an Online Member for zilch and go download Mac OS X Developer Tools including ProjectBuilder and InterfaceBuilder.

1. Please, open ProjectBuilder and start a new Cocoa Application.

I'm sorry, but I'm not going to be able to write "please" and "thank you" anymore when I ask you to do things. Alas, it's the ancient conflict: Barney vs. Word Count. Go to File > New Project…, a dialog asking you which kind of project you want to use appears. If the little blue triangle, called a "disclosure triangle," next to the word Applications is pointing to the right, click on it once to expose the subcategories. Click on Cocoa Application.

You will then be prompted to put a name in the upper text box, you can use "MyFirstProject" if something more brilliant doesn't come to mind. Notice that you don't need to put an extension (dot-something) because the title is that of a folder created by ProjectBuilder. You may use the Set… button to put this folder in a location other than the one it is proposing in the second text box. The Set… button causes a navigation panel to appear and once you've highlighted the folder into which you want your project folder to go, click on the Open button. Click Finish in the naming dialog.

David Trevas lives in Houston, Texas with his wife and daughter, two dogs and two Macs.

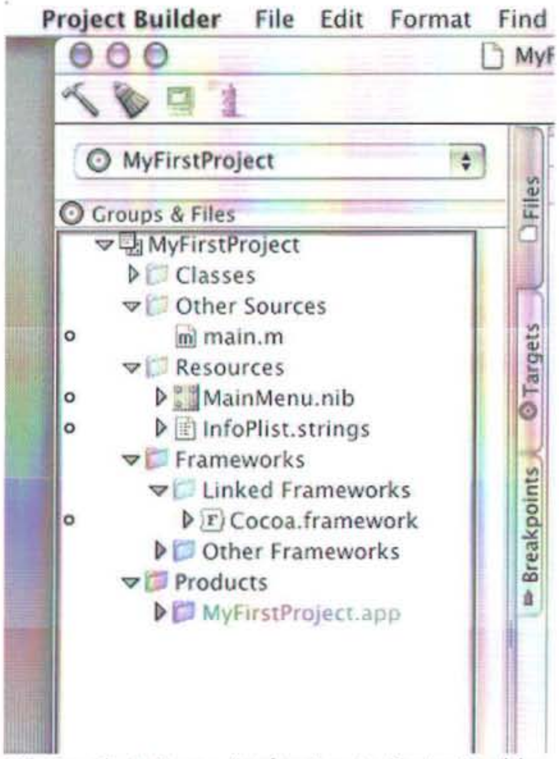


Figure 1. *A Cocoa Application in ProjectBuilder.*

Your project then opens up and you'll see a column on the left with five headings each flanked on its left by right-pointing disclosure triangles (see **Figure 1**). The first is Classes, it is where the code for the building blocks of the project should go. The second heading is Other Sources and it contains main.m, which is where everything really happens once the program starts to run. Happily, you don't need to touch this file for this project and you may get through many more projects before you become concerned with the details of its function. The third heading, Resources, is the one in which you will be working mostly. It contains MainMenu.nib which is the basis of the user interface we will be creating. The fourth heading is Frameworks and under the subheading Linked Frameworks resides Cocoa.framework which connects us with the years of work done at NeXT and Apple that makes our lives easier. Finally, the last heading called Products contains MyFirstProject.app which is end result of our labors, the executable file (Psst! Most normal users will see this as a single file, when, in fact it is really a package containing several different files. Just thought you'd like to know!)

2. Double-click the MainMenu.nib to open InterfaceBuilder.

The first thing to notice is that by double-clicking a file in ProjectBuilder, you open up InterfaceBuilder. The two are actually separate programs (applications) that are conveniently aware of one another. The next thing you will notice, or be overwhelmed by, is that your simple double-click has caused a great many things to happen. Overlaying you ProjectBuilder window are now four windows entitled: "MainMenu.nib…", "Cocoa - (something) Palette", "MainMenu - MainMenu", and "My Window" You might find it a little neater to go to the InterfaceBuilder menu and Hide Others to reduce the clutter.

A little side note: Cocoa is based on work done at NeXT which Apple bought in 1997. You will see that heritage show through time and again. For instance, a nib file is an acronym for NeXT Interface Builder (and I'll bet that it is not merely coincidental that a nib is the tip of a pen). You will also see the prefix "NS" on all of the objects which stands for NeXTStep.

Design the User Interface

From your perspective as one who uses programs, you may recognize the things you see in the palette as the places you put data or ask for something to happen or see the results of the program's "thinking." Each element is an "object" (a concept I will explain later) that has already been written and this frees you from a lot of code-writing. "A lot of code-writing" often means "tedium with a big probability of making mistakes."

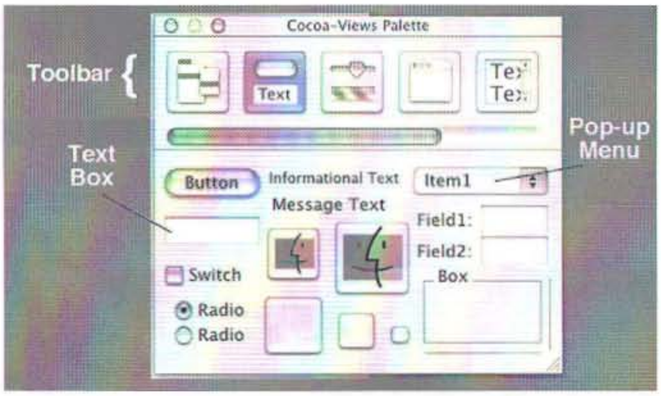


Figure 2. *The Views Palette*

Click the icons on the toolbar at the top of the "Cocoa - (something) Palette" window until you get to the one that says "Cocoa - Views Palette" and looks like **Figure 2**. A little time-out here is necessary to clarify some of this AppKit jargon. First, what the heck is AppKit? Cocoa has two main aspects: Foundation takes care of things related to the fundamentals of programming, while AppKit (short for Application Kit) helps you with things pertaining to applications such as windows, buttons and menus. Now, what's the deal with views? As one who works a lot with CAD, I normally associate views to things like Front View, side View, etc., but that isn't correct here. Loosely speaking, if you substitute "visible thing" for "view", you'll be good to go at this point. Among these visible things are those items referred to as user interface elements, controls or even gadgets.

3. Add user interface elements to "My Window".

Here you will add several of the typical Macintosh user interface elements, termed views by InterfaceBuilder. If you complete this entire section, you should end up with something like **Figure 3**.

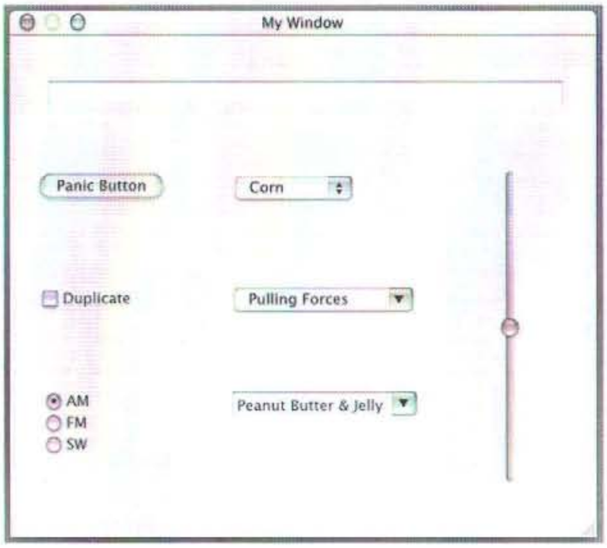


Figure 3. *Window Full of Views (Visible Things).*

Text Fields. Click and hold the mouse over the white box called the text box (or text field). Drag it into the window entitled "My Window" and release it. After you release the text field in the window, little bubbles appear around its perimeter. Grab the middle one on the right-hand edge and pull it toward the other edge of the window and you now have a much wider box.

Go to File>Save (or type Cmd-S) to save the nib file. Try to get in a habit of saving often - I have found InterfaceBuilder to be, charitably speaking, a bit quirky. Better save than sorry!

Buttons. Drag the Button from the palette to My Window. Double-click on the word "Button" to edit it. This theme of double-clicking on words to edit them is fairly consistent in InterfaceBuilder. I put the words "Panic Button" there and appreciated the automatic resizing. Did you?

At this point, you have a choice: add more things and explore the subtleties of each or get in the Express Lane and jump to the next section. For the rest of this section, in fact, you can skip down without any long-term adverse effects, assuming that you come back later and do this exercise completely.

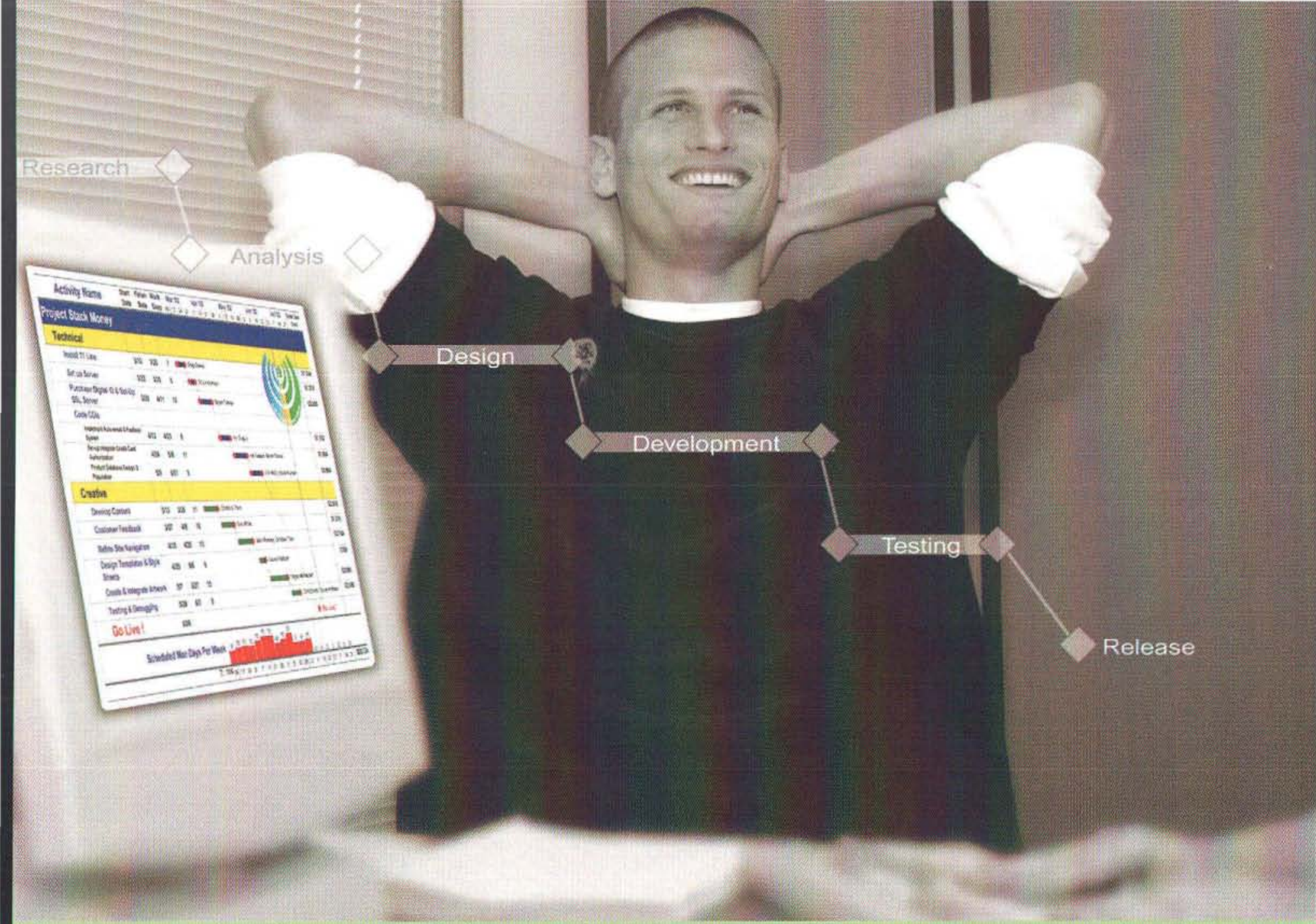
Check Boxes. Another easy view (visible thing) is the check box, which is called "Switch" on the palette. Drag a switch from the palette and put it in the window. Since we're talking about checks, I label the check box "Duplicate" to remind you of the option of having instant copies so you don't have to balance your checkbook in front of me in the grocery store line. Thanks for bringing your 23 items to the express line, too! Did you remember how to edit the text? That's right, double-click on the word "Switch."

Radio Buttons. Turning it up a notch, we will move on to radio buttons and because their behavior is that "one and only one" is activated, they only make sense when more than one is present. Drag a Radio into the window and notice that you get two for the price of one. Now, you say, "Your window has three radio buttons, but I only have two and if I repeat the step I'll have four buttons, which is still not three, so, what the heck?!" Ah, yes, I remember the feeling well and I must confess, I didn't use "heck". But the I found Alt-dragging (a. k. a. Option-dragging) and all was well. Click once in the midst of the radio buttons and a box with little dots around it appears. Hold down the Option key and grab the bottom center dot and drag it down–perfectly aligned radio buttons materialize like magic. By the way, had you selected the lower-right dot and dragged down and to the right, you'd get rows and columns of radio buttons.

Radio 1, Radio 2 and Radio 3 are not exactly names that apply for every situation so now you need to edit the words. Double-click on "Radio 1" and sigh as your editing ability hasn't returned. Time to call the Inspector! Click once in the empty area of "My Window" to undo this last selection. Press Cmd-Shift-I (or go to Tools>Inspector, but you will probably be glad to learn the keyboard shortcut) and look at the new window that appears, the title bar tells you what kind of object you are inspecting. Push it off to the right a little so you have a better view of the radio buttons. Click once on the buttons and a frame surrounding all three should appear. The title bar tells you that you are looking at an NSMatrix, that is, an array of objects. Double-click on Radio 1 again and now the Inspector title bar says that your are looking at an NSButtonCell. Double-click now on the button cell and you can edit it. I called them "AM", "FM" and "SW" for short wave since these could be buttons on a radio. If you double-clicked on the Matrix and Keanu Reeves and Lawrence Fishburne popped out and started kung-fu fighting, you clicked the wrong Matrix. The difference between an NSButton and NSButtonCell is not important for this project, but feel free to poke around in the AppKit documentation to find out more. Sorry, no extra credit.

Pop-Up Menus. These are straightforward, but things get a little trickier for pull-downs. Drag a pop-up menu from the palette to "My Window". Double-click over the phrase, "Item 1", and you will notice that three items appear. What if you want more? Keep in mind that this is a menu and what you will learn in the next section about menus will apply. Double-click again over an item's words to be able to edit them. I called my items, "Corn", "Toast" and "Weasel" for things that go "pop."

Pull-Down Menus. The way I found to create a



One incredibly successful project manager.

What does this guy know about software development that you don't? He knows that with FastTrack Schedule 7.0 he'll shave hours off the time it takes to organize, track and manage the details of all his development projects. And his eye-popping schedules are sure to turn heads and get results.

Whether the project on tap is coding the next killer app or putting a fresh coat of paint on a client's website, FastTrack Schedule's three powerful views display your information the way you want—as a schedule, a calendar, or as a resource graph that tracks the people, equipment, and materials crucial to project success.

And with our new compatible Palm OS version, you can sync schedules between your desktop and handheld. So even when you're on the run, your schedules are right at your fingertips. FastTrack Schedule also includes FastSteps™ intra-application scripting, full support for AppleScript, and is available in a compatible Windows version. For a free demo version or to order, call us today at **800.450.1983** or visit **www.aecsoft.com**.

Easily** the best in **Project Scheduling!

AEC SOFTWARE FastTrack Schedule Version 7 FastTrack Schedule for Palm OS Details

finalist macworld eddys

MVP Winner PC Computing

5-STAR PC Computing

pull-down menu is to start its life as a pop-up. Again, drag a pop-up menu from the palette to "My Window". Double-click over the phrase, "Item 1", and double-click it again to be able to edit it. In a pull-down menu, the first item is not a choice but the permanent title of the menu, just like a menu title from the menu bar. I'm calling this menu "Pulling Forces" and hitting Return. Call up the Inspector (Cmd-Shift-I or Tools>Inspector...) and click once anywhere in the empty space of the window and return to the pop-up you want to change and click it. The Inspector's title bar should read "NSPopUpButton Inspector" and if it is not already showing, choose "Attributes" from the pop-up just below the title. Simply choose the PullDown radio button to do the switch. Now, when you double-click over "Pulling Forces", you will see Items 2 and 3 in a box slightly below the button itself. Double-click on Item 2 and call it "Gravity" and change Item 3 to "Magnetism".

If you've done this kind of work before, you will notice some things missing in the Views Palette. That's why there is a More Views Palette (see **Figure 4**). From this palette, we will take two elements: a slider and a combo box.

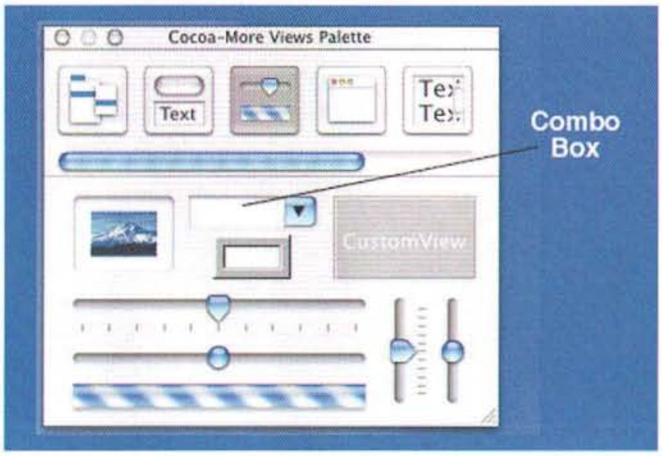
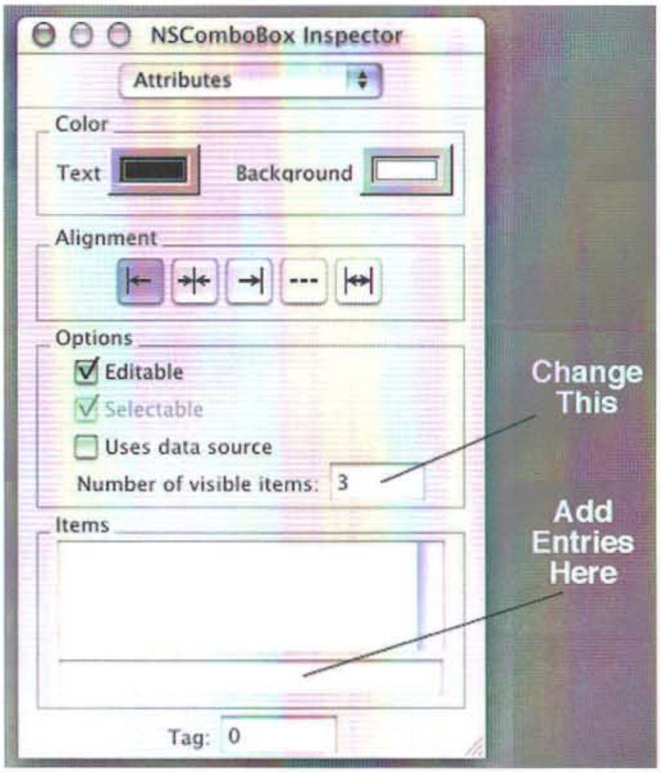


Figure 4. *The More Views Palette.*

Sliders. While you see four different sliders, they are all the same except that their attributes are set differently. Choose the one you like best and drag it on to "My Window." You can play with the attributes if you like. Of course, how am I going to stop you if I didn't want you to play with them?

Combo Boxes. Finally, drag a combo box from the More Views palette into "My Window." It is a somewhat a combination of a text box and a pop-up menu.



☑ Editable
☑ Selectable
☐ Uses data source
Number of visible items: 3
Tag: 0

Figure 5. *NSComboBox Inspector.*

To put in your items, call up the Inspector (Cmd-Shift-I, or Tools>Inspector, if you insist) and choose "Attributes" if you are not there already. The entry box is that little one at the bottom (as seen in **Figure 5**). Click in that box and type the text you want and hit Return. Repeat as often as you wish. I entered "Peanut Butter & Jelly" , "Macaroni & Cheese", and "Soup & Salad." I also changed the "Number of visible items" to three.

Feature: The Macintosh Human Interface Guidelines.

We're plopping the views (visible things) somewhat chaotically in the window and we'll be asking some of them to behave in ways that are not normal in Mac programs. This is O.K. since we're just throwing together a little intro here. However, if you plan to create programs for wider consumption, there is some must-read material at the Apple web site that you just, uh, must read. If you love Macs, I'll bet that one big reason you do is the *Macintosh Human Interface Guidelines*. This book details the proper usage and placement of each element of the user interface and the fundamental Mac philosophy with regard to the interaction between person and computer. Since these guidelines have been widely available for years, software developers have been able to create programs with such consistent interfaces that learning to use them is much easier than on other platforms. So go download *Macintosh Human Interface Guidelines* and don't forget *Mac OS 8 Human Interface Guidelines* and *Aqua Human Interface Guidelines* that add guidelines for elements that were introduced in Mac OS 8 and Mac OS X, respectively.

4. Add a new menu title and items to that new menu.

Go to the Menus Palette. You'll notice that pre-defined menus for some of the most widely used menus are there. Then, there are three other items you use for creating your own menus and modifying them and the pre-defined ones: Submenu, Item and an apparently blank one.

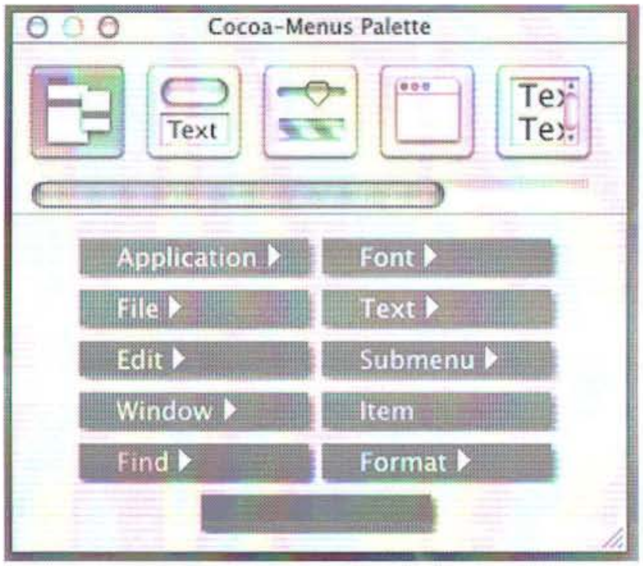


Figure 6. *The Menu Palette.*

"Submenu" means "menu title or menu item with lower level items"

"Item" is a menu item that doesn't have sub-items. Items are the only things on menus that interact with the rest of the project.

The blank one is a divider (not a uniter), also known as a separator. They simply provide a space so that you can group menu items into logical and visually appealing units.

When you complete this section your menus should appear like the ones in **Figure** 7.

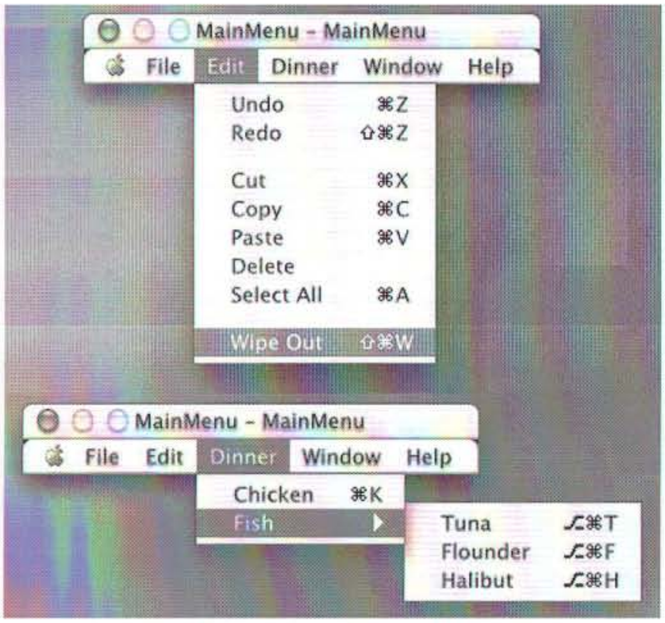


Figure 7. *The Menu System of the Project.*

Separators. The easiest thing to do is to place a separator. First, look at the window entitled "Main Menu - Main Menu" which might not sound like a

StoneTable

You thought it was **just** a replacement for the List Manager ?

We lied, it is **much** more !

Tired of always adding just one more feature to your LDEF or table code ? What do you need in your table ?

Pictures and Icons and Checkboxes ?
adjustable columns or rows ?
Titles for columns or rows ?
In-line editing of cell text ?
More than 32K of data ?
Color and styles ?
Sorting ?
More ??

How much longer does the list need to be to make it worth $200 of your time ?
See just how long the list is for StoneTable.

Make StoneTable part of your toolbox today !
Only $200.00 MasterCard & Visa accepted.

StoneTablet Publishing
More Info & demo Voice/FAX (503) 287-3424
http://www.teleport.com/~stack stack@teleport.com

DEV DEPOT℠

PowerBook Essentials!

KEYSPAN Digital Remote

The Keyspan Digital Media Remote is a powerful infrared remote which allows you to control multimedia applications on your computer in the same convenient way that you now control your home TV. Great for PowerPoint, QuickTime, DVD players, CD players, and MP3 Players!

These are products you need, they're **kool gadgets** you won't find anywhere else, you'll get more out of your Powerbook if you have these!

PCMCIA Card Ads 2 USB Ports

Just slide this PC Card into your notebook and, instantly, you can begin connecting USB peripherals.

KEYSPAN USB Twin Serial Adaptor

Add two serial ports to your USB Macintosh! Simple to use, attractive on your desktop, and reliable!

Neoprene Case for iBook

A perfect fit! Easy to slip your iBook in and out of this 1/4" (6.5mm) Neoprene iBook Glove.

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)

"World's Smallest AC Adapter" for almost all PowerBook and all iBooks!

AC Adapter for PowerBooks all G3s/iBook/3400

You spent how much to buy a portable computer? Then why do you haul a brick of a power adapter with you everywhere? This credit card sized power adapter makes your PowerBook truly mobile! Buy a second adapter for home!

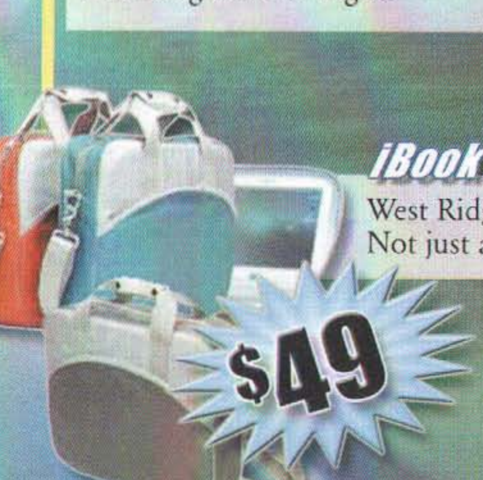
You spent thousands of dollars to buy a portable computer, and now you haul a power adapter brick with you everywhere, but a second adapter for the office, buy a small credit card sized adapter for travel, this thing is kool!

Auto Air 50W Adapter for all G3s/3400/iBooks

Power your iBook or PowerBook in a car or on a commercial airline flight. Forget swapping batteries, watch DVD after DVD on your next long drive or flight!

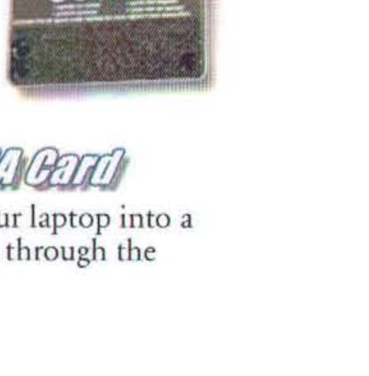
iBook Bag

West Ridge Designs introduces the "Macintosh inspired computer bag," the iBookBag. Not just another pretty face, the iBookBag is built to protect!



Cool Pad Rotating Laptop Stand

Keep your laptop Cool! The CoolPad helps dissipate heat and letsyou rotate or elevate your laptop for a perfect fit!



Farallon SkyLINE Wireless 11 mb PCMCIA Card

AirPort compatible, the SkyLINE PCMCIA card turns your laptop into a wireless wonder! Send eMail, surf the web, or transferfiles through the AirPort base station or other AirPort enabledlaptops!

www.devdepot.com/powerbook/

Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com weird name if you live on Bora Bora, eat couscous or have beri-beri. It represents the menu system of your project. Click once on the Edit menu of "Main Menu - Main Menu" to see a pre-written menu that has some of your trusty old friends like Cut and Paste on it. Meanwhile, back at the palettes, click on the menu palette icon at the far left of the toolbar. Click on the empty blue box at the bottom and drag it toward the edit menu. Notice that you are dragging a blue rectangle attached to the arrow representing the position of your mouse. It seems that InterfaceBuilder considers the tip of the arrow as your intention when moving something that takes up a lot of real estate. Move the tip of the arrow, therefore, to a point below the lower half of the words Select All and let go. A space should open up just below the Select All item. If it doesn't, hit the Delete key and try again.

Items. Once you've done this, you can put items in your own new section of the Edit menu. This time drag an "Item" from the palette to the Edit menu and drop it below the separator. To edit the text, we do what? Double-click on the word "Item" and type "Wipe Out" as if you wanted a command to nuke your current work so you can start over again.

If you are cruising through this exercise in the express lane, you can jump to the next step.

Command-Key Equivalents. Once you have become familiar with a program and tire of reaching for the menu bar for doing routine operations, you learn the command-key equivalents, also known as keyboard shortcuts. The key with the Apple and the propeller (or clover or flower) on it is called the Command Key (abbreviated Cmd in documents that don't have a font that has the symbol.) You know the typical command-key equivalents: Cmd-N for New..., Cmd-P for Print..., Cmd-X for Cut, Cmd-C for Copy, and so forth. To create a command-key equivalent for "Wipe Out", move your mouse to the Wipe Out item in the Edit menu and then to the far right side of that item, directly under Cmd-X and Cmd-C. Double-click and a little white outline rectangle that might hold one character appears. Here is where you type the keys that go with Command Key symbol (i.e., don't use the Command Key) which usually is a letter. Type "W" and nothing happens, why? Because, if you look at the File menu, you will see that Close is already assigned to Cmd-W. InterfaceBuilder automatically makes sure that you do not reuse keys or key combinations (of course, it would be nice to get a warning and then prompted to try something else.) Repeat the step to get the outline rectangle to appear and this time hold down the Shift key and type "W". You see that the characters Shift (an upward-pointing arrow), Cmd, W to the right of "Wipe Out" are now the command-key equivalent. Now you are ready to make your own menu.

Submenus. Drag a Submenu from the palette to the Main Menu bar and place your cursor arrow between Edit and Window. Double-click the newly placed text and type in "Dinner". Click once on Dinner and underneath a single Item appears. Double-click on it and call it "Chicken." Now comes the most excruciating part of the project (at least it was for me–you may benefit from my ordeal.) To make the first command-key equivalent in a menu takes a keen eye and a steady hand. Ever so gently, place the tip of your cursor on the right-hand edge of the "n" of the word "Chicken," and double-click. If you got that rectangular outline to pop up on the first try, you definitely benefited from my tribulation. Use "K" for the keyboard shortcut. Is it just me, or doesn't it seem that the ability to create a keyboard shortcut should be available in the NSMenuItem Inspector window?

Now, for the more familiar use of submenu: a menu item with a right-pointing triangle that guides you to more choices. Drag a Submenu to the spot just below "Chicken" in the Dinner menu. Double-click it and call it "Fish." Click once on "Fish" to see the item and rename it "Tuna." Now, with you newly-found skill at placing the first keyboard shortcut on a menu, double-click at the right edge of the "a" in "Tuna". In that outline, hold down the Option key and type "T" and you will see that the keyboard shortcut for Tuna is weird symbol for Option, Cmd, T. (If any of you knows where that weird symbol for Option comes from, please tell me.)

Hopefully, you are now in tuna with the menu system and are not floundering about. If you are up to it, add an item below Tuna called "Flounder" and set its keyboard shortcut to Option-F (of course, the Cmd adds itself automatically.) And why not just one more for the halibut? Add an item and call it "Halibut" and set its shortcut to Option-H.

MAKE A NEW OBJECT IN INTERFACEBUILDER. Feature: Objective-C and Object-Oriented Programming.

Before we can continue, we need to discuss a few concepts and terms. Objective-C is a language that extends the C language by making the object the central unit of programming. It is the original language of Cocoa and its predecessors, although current Cocoa programmers have the option to do their work in Java. If you know C, you should be able to pick it up quickly by reading *Object-Oriented Programming and the Objective-C Language* which available freely from the Apple web site. If you don't know C, you can learn it later and just follow the code-writing parts as written. As one who has struggled several times to really "get" C++, I found that Objective-C provided a much better way to understand object-oriented concepts and yielded code that was much more readable (after I got used to the square brackets and the colons).

Cocoa applications are generally conceived as a network of objects. You can think of objects as people. Different objects know different facts (data) and possess different skills (methods). Like people, they can communicate with one another. Using messages, one object can ask another for information or to do something. They can also talk to themselves, but that is not a sign of mental illness for objects.

The descriptions of an object's knowledge and skills is contained in its class, also known as a factory in Objective-C. An object is an instance of a class (or the product created by a factory.)

A class can have children and pass along its data and methods to them. A child class can add more data and methods and alter the methods it inherited. The child class' methods can deviate from its parent's completely, slightly or not at all.

Often one finds a class that somewhat does what one wants and uses that as a basis to create an class that does exactly what one wants. Almost every class in a Cocoa project inherits (either directly or after a few generations) from a fundamental class called NSObject. NSObject handles things like object identification and memory management, so a great deal of knowledge and courage are required if you want to avoid it.

In this way of writing programs, your concept of data needs to expand from what you may have learned from earlier languages. As before, there is some data that lives within the object. For example, a Rectangle object may have data members called "Height" and "Width". In Objective-C, however, a common piece of data is just the address (or location) of another object. We say that this piece of data "points to" the other object. To return to the metaphor of people, it brings to mind the saying, "It's not what you know, it's who you know." Witness the ongoing value people put on their address storage systems from little black books to Rolodexes to organizers to Palm Pilots and you will see why there is power not only in knowing things and having skills, but also in knowing where to find others who have knowledge and skills that you don't.

5. Create a subclass of NSObject as the Application Controller.

We're going to need an object that acts as a traffic cop to direct all of the input that comes from our buttons, sliders, etc. and send some kind of output to the text field. We call this kind of object a "controller." We aren't yet aware of a good starting point for a controller, so we create the blueprint (or class) of a controller that inherits from NSObject. Another way to say it is that the controller should be a subclass of NSObject.

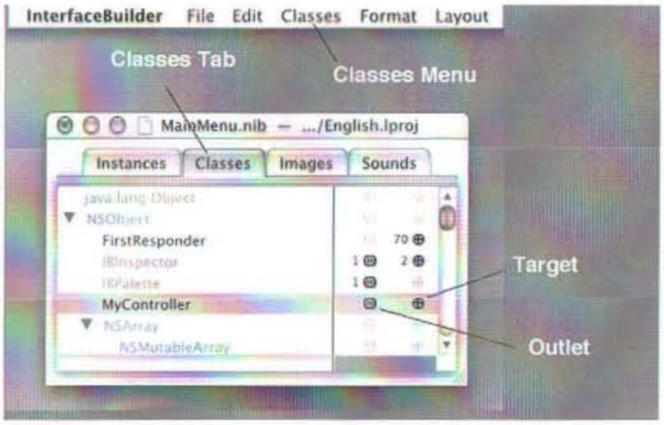


Figure 8. *MainMenu.nib Window with Classes Tab Selected.*

In the MainMenu.nib window, click on the tab called Classes and your screen should show you something very much like **Figure 8**. Scroll to the very top and find NSObject. Click the NSObject once and go to the Classes menu located on InterfaceBuilder's menu bar at the top of the screen. Select Subclass and a new class called "MyObject" appears, ready to be edited. Change the

Valentina
object-relational database engine

The fastest database engine for the MacOS

It operates 100's and sometimes a 1000 times faster than other systems

Valentina (AppleScript)(68K/PPC/X)	**$49**
Valentina C++ SDK (68K/PPC/X/Win32)	**$499/699**
Valentina for Java (PPC/X/Win32)	**$299**
Valentina for REALbasic (PPC/X/Win32)	**$199/299**
Valentina for Director (PPC/Win32)	**$199/299**
Valentina XCMD (PPC/Win32)	**$199/299**
Valentina for WebSiphon (PPC)	**$299**

Make your application's database operations blazingly fast!

Order Directly from Our Web Site **www.paradigmasoft.com** Hosted by MacServe.net

Download full featured evalution version

name to "MyController." (Using names beginning with "my" is an old cliche for introductory programming exercises. It is supposed to indicate the separation of the words that you must use exactly as specified by the language from the those with which you have some freedom. You may indulge my penchant for corniness or call this object something else if you want.)

6. Add an outlet to the text field.

An outlet can be thought of as the place where data goes when it is ready to leave the controller. To create one, click on MyController if it is not already highlighted. Go to the Classes menu at the top of the screen and choose Add Outlet. Call the outlet "theTextField."

Notice the two icons in the right-hand column. The first one with the two vertical lines is supposed to remind you of an electrical outlet. Since you added the outlet, the number 1 appears just to the outlet symbol's left. And, by the way, this is one of those pieces of data that is only the location, or address, of another object. In an upcoming step, you will be tell this piece of data the object for which you want to store the address.

7. Add actions for each control and menu item.

The other symbol next to the outlet is a small gray circle with crosshairs and it is called a target. These are to what the buttons, sliders, menu items and so forth will be shooting their requests. Instead of being the address of an object like the outlet is, an action occurs when the target is hit. An action is a special type of method.

Highlight MyController again if it isn't still in focus. Go back to the Classes menu again and this time choose Add Action. Start typing your action name and it will replace the generic myAction: that appeared. Make sure that you end each name with a colon.

Here are the ones I used:

```
pushPanicButton:
checkDuplicates:
radioAM:
radioFM:
radioSW:
popCorn:
popCork:
popWeasel:
pullGravity:
pullMagnetism:
slideHeadFirst:
comboLunch:
menuEditWipeOut:
menuDinnerChicken:
menuDinnerTuna:
menuDinnerFlounder:
menuDinnerHalibut:
```

Those of you in the express lane need only create the pushPanicButton: and menuEditWipeOut: actions.

8. Instantiate the new controller.

What you've done in the previous three steps is create a class (or factory) for a controller much like an architect creates a blueprint for a house. Can you live in a blueprint? No, you have to build a house according to that blueprint to have something to inhabit. Similarly, you have to create an instance of the controller class in order to use it in your program. The verb, "Instantiate", was invented in 1949, according to my dictionary, to compress the phrase "create an instance" into a single word.

Now that the concept is clear in your mind, go to the Classes menu and choose Instantiate. Click on the Instances tab of the MainMenu.nib window and you'll see that there's good news and there's bad news (see **Figure 9**).

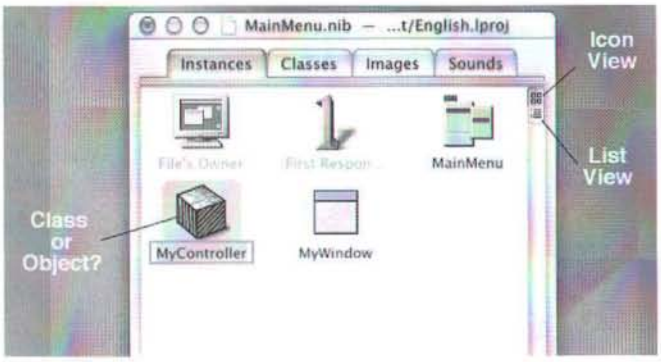


Figure 9. *The MainMenu.nib Window with Instances Tab Selected.*

The good news is now you see a cube labeled "MyController" in the window (provided you are in icon view. If you are in list view, look at the far right side of the window at the top of the scrolling area, and you will see a small square with squares in it. That is the icon-view button. The square below it with the horizontal lines will get you back to list view. Now you're marveling at the cube you've created and are wondering what could be bad about this. In informal discussions, we often use the terms "class" and "object" interchangeably, when, in fact, an object is an instance of a class. To return to the metaphor of people, a class is like a job description and an object is like the living, breathing person who has the job. The MyController you have built from steps 5–7 is a class and the one you just instantiated here is an object of that class. The convention in Objective-C is to capitalize the first letter of a class, and to have an object start with a lower-case letter. Double-click on the words "MyController" and rename it "itsMyController." Now, your joy can be unbridled!

CONNECT THE CONTROLLER TO THE VIEWS (VISIBLE THINGS).

Now that the cube representing the controller appears in the MainMenu.nib window, you have a visual representation

Installer VISE 7.3

VISE

Build one installer for

System 7.1 through Mac OS X.

www.mindvision.com

of all of the objects in your project. Perhaps you think it might be pretty cool if you could connect them all up with some points and clicks of the mouse. Well, you can! The basic principle is that you click on the "from" item, hold down the Control key, drag to the "to" item and let go.

9. Connect the controller to the text field.

For an outlet, you want data to flow from the controller to the outlet object. Click once on the icon called "itsMyController" to select it. Now, hold down the Control key and drag the L-shaped line to the text field. This is shown in **Figure 10**. A dialog box opens up an allows you to select an outlet. Choose theTextField and then click on Connect.

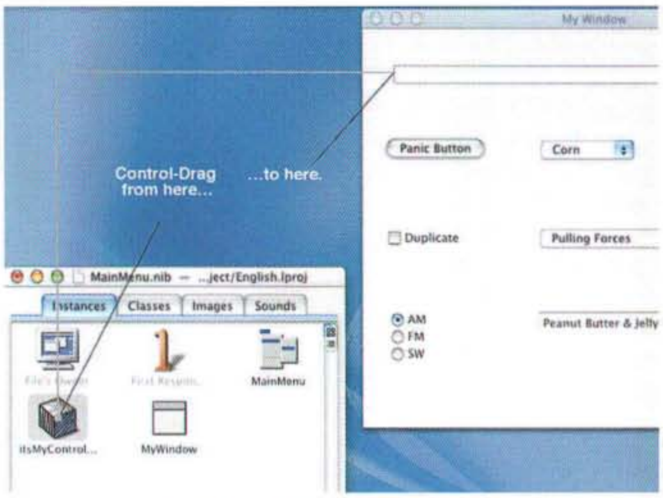


***Figure 10.** Making the Connection.*

10. Connect the controls and menu items to the controller and select the right actions.

Fortunately, connecting views (visible things) to the controller works the same way. This time the request for action comes from the view (visible thing) and goes to the controller Select, say, a button, then Control-drag to the "itsMyController" icon. The difference here is that you are now prompted for an action. This means choosing the radio buttons individually (as NSButtonCells) to connect each one with the right target. Likewise with the pop-up and pull-down menus, except it is NSMenuItems you must connect individually.

In the dialog that pops up after you've made a graphical connection, the actions you may connect to are listed in the right-hand column. If not, click on target in the left-hand column.

Menu items behave exactly the same way, so all you have to do is Control-drag from the menu item to the controller in order to connect the menu item with the action it should do.

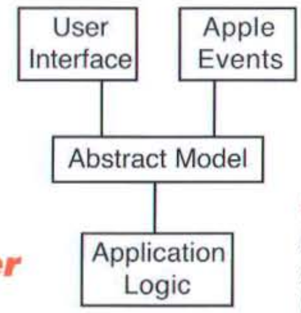
Generate the Code and Write the One Line.

11. Create the skeleton of the code for the controller.

When you've completed this work, a little treat appears. Since you've been doing repetitive connections for a while, you probably are not looking forward to writing code for all of those actions. The good news is: you don't have to. Simply click once on the "MyController" icon and go to Classes>Create files... and navigate to the folder of this project. Ba-da-boom, ba-da-bing! The basic interface file

Fight Boredom

Let's face it: Much of programming is boring and repetitive. Well, that's where the right tool can save days, weeks, or even months of your valuable time.



Model-View-Controller

AppMaker's generated code uses the MVC paradigm. It separates the user interface from application logic, making code easier to write. You deal only with abstract data; AppMaker takes care of the user interface.

AppMaker

Your Assistant Programmer

AppMaker makes it faster and easier to make an application. It's like having your own assistant programmer. You point and click to tell AppMaker the results you want, then it generates "human, professional quality code" to implement your design.

Scriptable Applications

AppMaker generates the 'aete' resource and generates code to access your data (Properties and Elements in the Apple Event Object Model) and to handle Events.

Just $199 from www.devdepot.com

B•O•W•E•R•S
Development

P.O. Box 929, Grantham, NH 03753 • (603) 863-0945 • FAX 863-3857
bowersdev@aol.com • http://members.aol.com/bowersdev

(ending in .h and called a header file in C) and its implementation file (ending in .m, the Objective-C source file ending analogous to .c in C and .cpp in C++) appear in magically ProjectBuilder.

You can now save your nib file and quit InterfaceBuilder now that your interface is, uh, built. Click once on the ProjectBuilder icon in the Dock to bring it out of hiding. The version of ProjectBuilder/InterfaceBuilder that I am currently using puts the MyController.m and MyController.h files in strange places. Please put them in the top folder, the Classes folder. Thank you.

The concept of separate files for interface and implementation makes a lot more sense as projects get bigger and bigger. It is a better idea, therefore, to get you into good habits when your starting off small than trying to force change on you after you've developed bad ones and are entangled in a huge mess. The interface is the place where you introduce the data (variables, members) and show what methods (functions, subroutines) are available to the object. The implementation is where you put the mojo, the detailed instructions of how to actually do the things you promise in the interface.

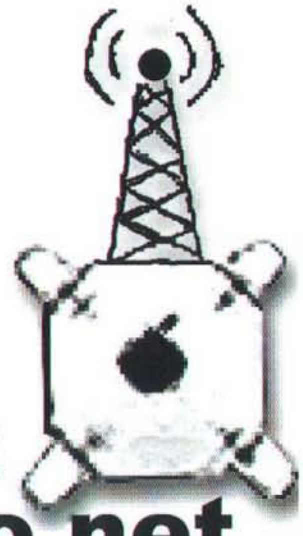
Here is the code generated for you and placed in MyController.h. After you examine it, I will explain some of the features of this code that are unique to it and not Objective-C concepts you can get by reading Chapter 3 of the book mentioned above.

Listing 1: The interface of the controller class.

```
#import <Cocoa/Cocoa.h>

@interface MyController : NSObject
{
  IBOutlet id theTextField;
}
- (IBAction)checkDuplicates:(id)sender;
- (IBAction)comboLunch:(id)sender;
- (IBAction)menuDinnerChicken:(id)sender;
- (IBAction)menuDinnerFlounder:(id)sender;
- (IBAction)menuDinnerHalibut:(id)sender;
- (IBAction)menuDinnerTuna:(id)sender;
- (IBAction)menuEditWipeOut:(id)sender;
- (IBAction)popCorn:(id)sender;
- (IBAction)popToast:(id)sender;
- (IBAction)popWeasel:(id)sender;
- (IBAction)pullGravity:(id)sender;
- (IBAction)pullMagnetism:(id)sender;
- (IBAction)pushPanicButton:(id)sender;
- (IBAction)radioAM:(id)sender;
- (IBAction)radioFM:(id)sender;
- (IBAction)radioSW:(id)sender;
- (IBAction)slideHeadFirst:(id)sender;
@end
```

"IBOutlet" means nothing, it is just there to remind you of the purpose of the object there. You may be offended by this apparent waste of space, but remember, code is not just a communication between you and the machine, it is a record of your intended goals for others to read (and one of them can be you months or years later.)

"IBAction" means "void" which, in the case of a method,



MacOS
Radio.net

www.macosradio.net
(a Mardun Software Ltd. company)

•Talk radio about the Mac, only on the Internet.

•No more surfing to 100 different sites to get your MacOS news.

•Interesting guests from companies you care about and sources you listen to.

ALL MAC
ALL THE TIME.

For more information about Internet radio, QuickTime Streaming Servers and how you can develop the power of MacOS X Server for media streaming today, e-mail sales@macosradio.net or call 573-443-0301

Mardun

©2001 Mardun Software Ltd.

states that it returns no value.

"sender" refers to the view (visible thing) that is causing the action. As you progress, you may be interested in knowing the value to which the slider is pointing, for instance, and then you will make use of the sender object.

Here is a question you may ask: O.K., wise guy, you said that each user interface element was an object, but I only see one, "MyController", where are the others? Good question, although I could do without the attitude. The views (visible things) are stored in the nib file — the construction and functional details of "My Window" and everything in it is kept in the nib file. To keep you on the edge of your seat, notice that the controller is different from the views (visible things). Hmmmmm?

12. Add this code to any of the actions.

Go to the MyController.m file and place your cursor between the curly braces of the first method, checkDuplicates().

Now, the moment you have all been waiting for (or dreading): Here is the one line of code you need to make this project do something.

```
[theTextField setStringValue:@"You've done this
action."];
```

How did I come up with that? When I put the text field in "My Window" is noticed that its Inspector indicated that it was an NSTextField. I went to the Application Kit Table of Contents page (AppKitTOC.html) which came with the Developer Tools, but is also at Apple's web site. I clicked on NSTextField and scanned the available methods and did not find a suitable one. I remembered that classes inherit the methods of their parents. At the top of the page, I saw that NSTextField is a child of NSControl (the lineage is expressed with the closest ancestor being the leftmost in the chain.) On the page for NSControl, I found the setStringValue: method which was exactly what I needed.

Listing 2. The top lines of an early version of the implementation of the controller class.

```
#import "MyController.h"

@implementation MyController

- (IBAction)checkDuplicates:(id)sender
{
  [theTextField setStringValue:@"You've done this
action."];
}
```

Here, the square brackets indicate that enclosed text is a "message" where one object tells another to do something.

The message has two major parts: the first is the receiver (in this case, theTextField). It is the object that receives the request from the bossy controller.

The second part is the method (setStringValue:) and it represents the method or function for the receiver to perform. The method must be one that the receiver can do or you'll get an error.

The colon ":" is significant because the number of colons in a method definition indicates how many parameters, or arguments, that the method requires. Here, the one colon means that one parameter is required for this method and it just so happens that it needs to be an NSString. A string is a group of letters, numbers and other characters. However, you want to make sure that the compiler (the program that translates the code you written as text that someone could read into instructions that a computer can read) does not think your string is part of a computer language or a name of a class, variable, method, etc. In C, one simply puts a string between double quotes. An NSString offers many advantages and is most easily created by putting an at-sign ("@") before the string you enclose in double quotes.

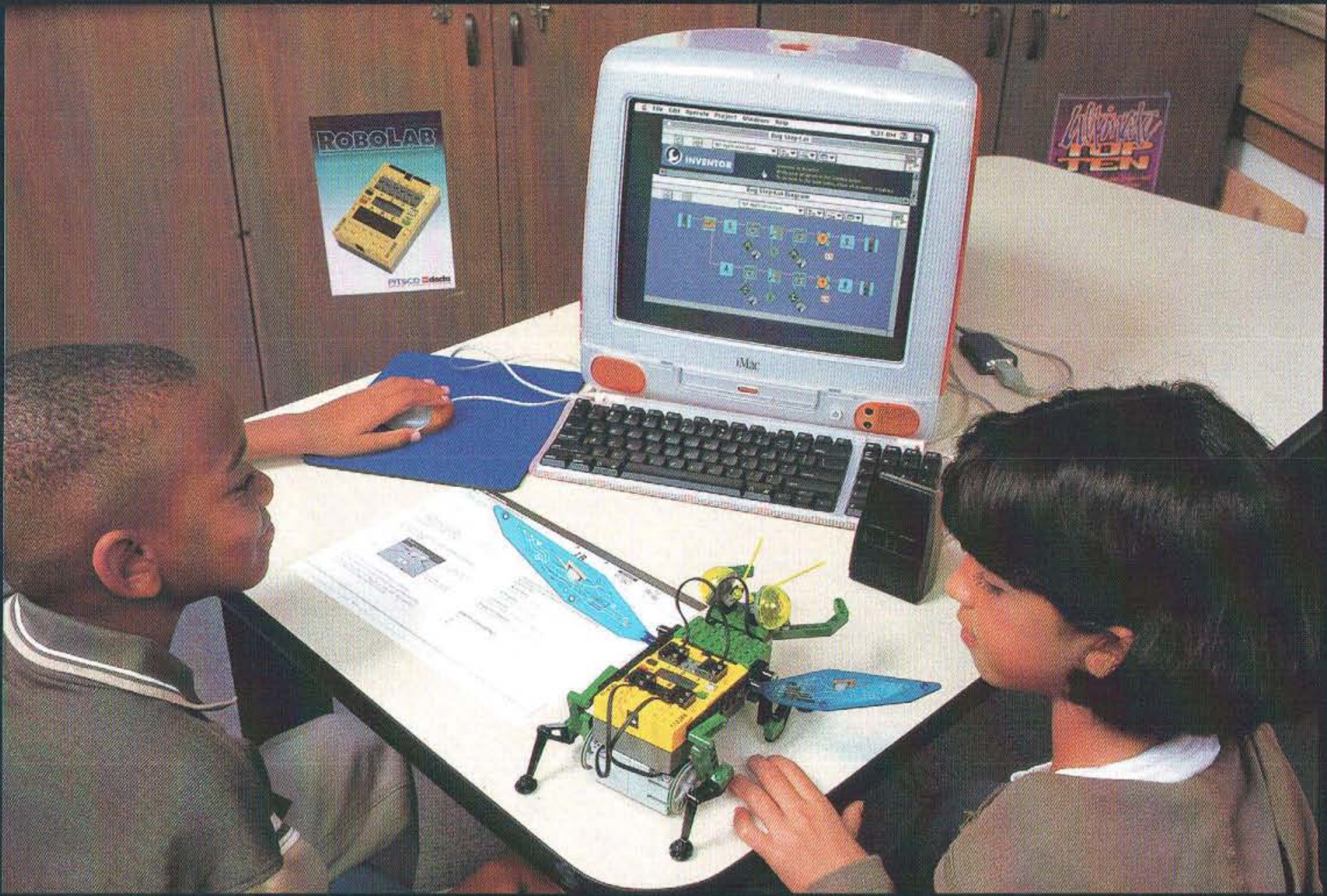
Don't forget the semi-colon after the message as that signifies that the "line" of code is complete. A "line" of code in any of the C-based languages can span several lines of screen (or paper) and often improves clarity. For that capability, we accept the responsibility of explicitly terminating every line of code with a semi-colon. If you are new to programming, you will probably make this error repeatedly, but fortunately, most compilers these days are smart enough to show you where you may have forgotten to put a semi-colon.

Feature: The Model-View-Controller Scheme.

There is a method of creating large projects by creating three different types of objects. This being a small project you've only worked with two out of three (and Meat Loaf fans would say that ain't bad.) You have used the views (visible things) provided by InterfaceBuilder and dealt with them using the controller you created. In the controller, you wrote the single line of code to take the input and generate an output, but you can imagine that you may want to send the input to an object that could some fancy stuff with it and gives you back some amazing result that the controller could pass back into the views. These objects that work the really interesting stuff on the data are called models. Creating a project in this manner observes the model-view-controller scheme.

There are many benefits to doing your work this way. The models and views, particularly custom views you may develop later on, are reusable in various projects. It makes a large application easier to maintain by allowing tweaking of models or views without having to worry about the other – the controller or controllers provide the separation to ensure this. This means, of course, that the controllers are very specialized for each project and are generally not reusable.

Some projects may get a performance boost if you



LEGO MINDSTORMS™ FOR SCHOOLS

Hands On Learning @ Home
handsonlearning.com

Stimulate early design, engineering and computing skills with these challenging materials. The MINDSTORMS for Schools Team Challenge Building Set enables students to simulate real-life robotics and robotic systems. The set contains 715 items, including an RCX brick, an infrared transmitter with a cable for a PC or Mac, motors, a light sensor, touch sensors, building instructions, and a sturdy plastic storage container. The icon-based ROBOLAB™ software, appropriate for children ages 8-16, is sold separately. The set is supported by two activity packs that stimulate open-ended problem solving.

PITSCO LEGO dacta
PARTNERS IN INNOVATIVE EDUCATION

www.pldstore.com•800-362-4308
www.pitsco-legodacta.com/products/mac.htm

combine the functionality of controllers with either models or views, but that probably won't be a burning issue for most of us. The model-view-controller method is an excellent way to structure a project, but I wish it were called model-controller-view to emphasize the separation of the model and view by the controller.

13. Copy and paste this line to all of the other actions.

Highlight the message you just wrote and choose Edit>Copy (Cmd-C). Paste (Cmd-V) this line between all of the action methods in the implementation file, MyController.m. Pasting copies of this function does not really count as writing, so the title is still truthful, right?

14. Allow user to modify the string to be more useful.

I also didn't say you wouldn't have to *modify* that single line. As we discussed above, a string is a group of characters that were not part of a programming language. If you are modifying text that is not as restricted as actual code, this activity shouldn't count as writing code. You don't really have to do this, but your application will be awfully boring if every time you do something, the generic message always shows up.

Listing 3: The final version of the implementation of the controller class.

```
#import "MyController.h"

@implementation MyController

- (IBAction)checkDuplicates:(id)sender
{
  [theTextField setStringValue:@"You chose duplicate checks."];
}

- (IBAction)comboLunch:(id)sender
{
  [theTextField setStringValue:@"You have chosen your lunch
combo."];
}

- (IBAction)menuDinnerChicken:(id)sender
{
  [theTextField setStringValue:@"You selected chicken for
dinner."];
}

- (IBAction)menuDinnerFlounder:(id)sender
{
  [theTextField setStringValue:@"You selected flounder for
dinner."];
}

- (IBAction)menuDinnerHalibut:(id)sender
{
  [theTextField setStringValue:@"You selected halibut for
dinner."];
}

- (IBAction)menuDinnerTuna:(id)sender
{
  [theTextField setStringValue:@"You selected tuna for
dinner."];
}

- (IBAction)menuEditWipeOut:(id)sender
{
  [theTextField setStringValue:@"You want to wipe everything
out?"];
}

- (IBAction)popCorn:(id)sender
{
  [theTextField setStringValue:@"You have popped corn."];
}

- (IBAction)popToast:(id)sender
{
  [theTextField setStringValue:@"You have popped toast."];
}

- (IBAction)popWeasel:(id)sender
{
  [theTextField setStringValue:@"Pop goes the weasel."];
}

- (IBAction)pullGravity:(id)sender
{
  [theTextField setStringValue:@"Gravity pulls all."];
}

- (IBAction)pullMagnetism:(id)sender
{
  [theTextField setStringValue:@"Magnetism pulls iron."];
}

- (IBAction)pushPanicButton:(id)sender
{
  [theTextField setStringValue:@"Calm down, all will be
fine."];
}

- (IBAction)radioAM:(id)sender
{
  [theTextField setStringValue:@"AM radio - traffic, weather
and news"];
}

- (IBAction)radioFM:(id)sender
{
  [theTextField setStringValue:@"FM radio - all hits,
all day, all night."];
}
```

high quality - competitive rates - 16 years experience - award winning

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

Full Spectrum Software

Development & Testing

Device Drivers

Porting **Plug-ins**

TCP/IP

Carbon / OSX

Cross Platform Development

One Bridge Street
Newton, MA 02458

617-965-0029
www.FullSpectrumSoftware.com

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

competitive rates - 16 years experience - award winning - high quality

```
- (IBAction)radioSW:(id)sender
{
  [theTextField setStringValue:@"Short Wave - Ham it up!"];
}
- (IBAction)slideHeadFirst:(id)sender
{
  [theTextField setStringValue:@"Safe at home - we win!"];
}
@end
```

Bring the Project to Life

15. Build and run it.

Simply go to the Build Menu and select "Build and Run." If errors occur, double-check the code you wrote in MyController.m to ensure that the curly braces, the square brackets and the quotes all have matches. Make sure that each string in quotes has an at-sign in front of it and that there is a semi-colon after each closing square bracket.

If everything goes well, on the other hand, you may be disappointed that nothing has apparently happened except that the menu bar reflects your changes. Just go to the Window menu and select "Bring All to Front". The reason you needed to do this manually is because I did not discuss the lines of code you would need to bring the window to the front on startup. You should now see you creation in all of its glory!

Click the buttons, slide the slider, choose each item in the pop-up and pull-down menus, click on your menu entries and try the keyboard shortcuts. Did they all work as planned? If so, congratulations, otherwise you may notice things you forgot to connect or connected in correctly.

When you are finished, you can click Cmd-Q or go to MyFirstProject>Quit to quit your application.

Now, was that fun or what?!

Conclusion

If you are reading this, you have covered a lot of ground in a short time and you should be proud of yourself. You've learned how to use ProjectBuilder and InterfaceBuilder and were introduced to objects and classes, Objective-C, inheritance, messages, views (visible things), menu making, target/action, outlets, connections between objects, model-view-controller, human interface guidelines, AppKit and Foundation (which make up the Cocoa framework), methods, receivers, interface and implementation, inspectors, keyboard shortcuts, instantiating and subclassing (I hope I haven't missed anyone.) By touching on these topics, I hope you got a good "big picture" view of what Cocoa involves and are encouraged to pursue learning of this amazing way to making our Macs even more useful, friendly and fun. Best of luck and thanks for your time. MT

Webmasters! Start your web hosting business with edition.net reseller hosting!

- dedicated platforms
- single and multi-domain hosting
- your choice of Mac OS or Linux platforms
- FileMaker Pro hosting featuring Lasso
- QuickTime streaming server
- Our web-based control panels make it easy to
- deploy your multi-domain server

edition.net – the Fine Arts Network

http://www.edition.net • info@edition.net
+1 (877) 225-3821 toll free • +1 (310) 379-8563 international

DO YOU NEED TO "MODERNIZE" *YOUR* SEARCH TECHNOLOGY?

POWER TOOLS FOR YOUR WEBSITE:

Onix - Search and Retrieval Engine
RouteX - Document Filtering and Routing Engine
Brevity - Document Summarization Engine

http://www.lextek.com

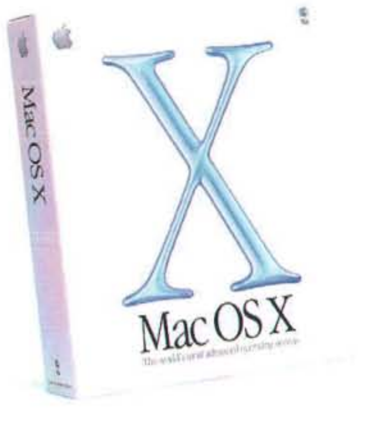
Internationalization and Localization

(Translation: Why you should you read this article.)

I'm not going to talk about localization or why you should have your application translated to French. Instead, I'm going to talk about something we both can relate to: money.

In 2000, almost half of Apple's revenues came from outside the U.S. So, if your application supports only U.S. English, you may be missing out on half of the market (and revenue) for your product. That's the bad news.

Here's the good news: With Mac OS X, Apple introduced new technologies to help you bring your application to international markets quickly and easily. Read on for more information and remember: If you don't get your product to foreign markets, you can always count on your competitors to do so.



Mac OS X: An International OS

With the introduction of Mac OS X, Apple redefined what a truly international platform should be. Take a look at a clean install of Mac OS X and you'll notice support for many languages—seven to be exact. At its introduction, Mac OS X was released in seven languages: English, Japanese, French, German, Italian, Spanish, and Dutch. A new version of Mac OS X was released during the 2001 Worldwide Developers Conference that contains support for Korean, simplified and traditional Chinese (as well as the rest of the European languages.)

Mac OS X streamlines internationalization. From a new International pane in the System Preferences, to new technologies that help developers take advantage of Unicode™, Mac OS X streamlines internationalization. A new delivery mechanism enables developers to ship several languages in one bundle. In addition, Mac OS X includes new technologies that handle Unicode and complex scripts. Both will be discussed in more detail here. These advents will undoubtedly make Mac OS X "Le international OS."

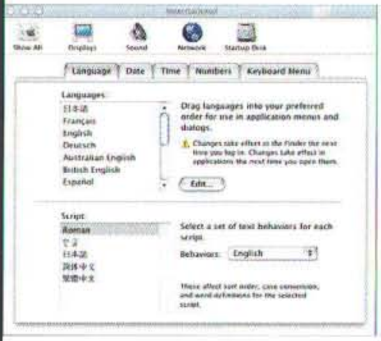
Internationalization Made Easy
(Translation: Why you should bundle your application.)

Mac OS X makes it easy to internationalize software and it does so in such a way that a single binary can support localizations for multiple languages and regional dialects. It also lets software developers dynamically add localized resources for new languages or regions. In Mac OS X, most software comes in the form of a bundle, of which an application package is just one type.

A bundle is an opaque directory in the file system that contains one or more executables and the resources that go with those executables. One of the primary benefits of bundles is the infrastructure they provide for localizing software. Localized resources such as image and string files, as well as Mac OS 9–style resources (rscs), can be put in bundle subdirectories whose names reflect a particular language or regional dialect (for example, Canadian French). A properly constructed Mac OS X application (or plug-in or shared library) does not hardwire paths to the resource files in these directories. Instead, when the application needs a resource, it uses a special system routine to obtain the localization that best matches the user's language preferences. (Please download "Inside Mac OS X: System Overview" if you'd like more information on Application Packaging and Bundles.). Go to:
http://developer.apple.com/macosx/

The new International pane in the System Preferences introduces users to a new paradigm in selection. In the following example (see fig. 1a), you'll see preferences regarding language ordering. The preferences were changed to have Japanese as the first language, French second, and English third. Next time the user logs in, the Finder will have its UI in Japanese. (Apple introduced

Xavier Legros has been an Apple employee for four years, three of which were working as an engineer on MLTE. Most recently, he joined the ranks of Apple's Technology Managers in Worldwide Developer Relations to promote Apple technologies. He regularly drinks champagne with developers each time they adopt one of his technologies...you could be next! He can be reached at xavier@apple.com.


Show All · Displays · Sound · Network · Startup Disk

Language · Date · Time · Numbers · Keyboard Menu

Languages:
日本語
Français
English
Deutsch
Australian English
British English
Español

Drag languages into your preferred order for use in application menus and dialogs.

Changes take effect in the Finder the next time you log in. Changes take effect in applications the next time you open them.

Edit...

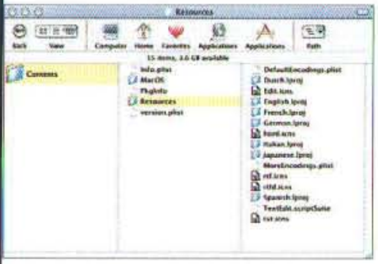
Script:
Roman
한글
日本語
简体中文
繁體中文

Select a set of text behaviors for each script.

Behaviors: English

These affect sort order, case conversion, and word definitions for the selected script.

Fig. 1a – This is the International pane from the System Preferences.


Resources

Back · View · Computer · Home · Favorites · Applications · Applications · Path

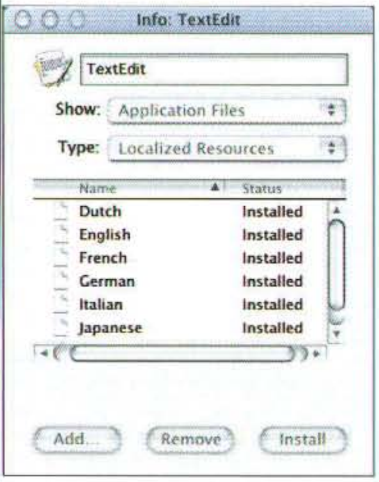
15 items, 3.6 GB available

Contents

Info.plist
MacOS
PkgInfo
Resources
version.plist

DefaultEncodings.plist
Dutch.lproj
Edit.icns
English.lproj
French.lproj
German.lproj
html.icns
Italian.lproj
Japanese.lproj
MoreEncodings.plist
rtf.icns
rtfd.icns
Spanish.lproj
TextEdit.scriptSuite
txt.icns

Fig. 2a – An example of what is inside a bundle. (TextEdit in this case.)


Info: TextEdit

TextEdit

Show: Application Files

Type: Localized Resources

Name	Status
Dutch	Installed
English	Installed
French	Installed
German	Installed
Italian	Installed
Japanese	Installed

Add... · Remove · Install

Fig. 3a – Localized languages installed as shown in the "Show Info" panel.

a new font called Hiragino with Mac OS X to display Japanese text. It looks gorgeous on the screen and in print.) The ordering of the languages means that if a user has an application that has been localized in French and English, the Finder will launch the application and load the resources in the French folder. From a developer's point of view, if you bundle your application and have the localized resources in the appropriate folders, then you don't have to do anything special in order to generate this behavior. The System will load the appropriate resources corresponding to the user's preferences.

As illustrated in fig. 2a, one can see that TextEdit has been localized in seven languages. The "MacOS" folder contains the binary and "Resources" contains all the localized resources (as well as some global resources like the "icns" files.) The file "info.plist" is the heart of the bundle. It contains the version of the application, the icon to be used for the documents the application creates, as well as the application icon and many other parameters specific to the application.

You can see the localized packages supported by the application through the Finder as well when you access the "Show Info" panel (CMD+I) (fig. 3a).

Text Support in Mac OS X: International by Default

Use CFString to Store Your Text

Core Foundation enables internationalization through Unicode strings and provides abstractions that contribute to operating system independence.

CFString and CFCharacter-Set provide a full suite of fast and efficient string manipulation and conversion functionality. String Services offer seamless Unicode support and thus greatly simplify internationalization. String Services also facilitate sharing of string data between Carbon and Cocoa applications.

Use MLTE for Your Text Editing Needs in Carbon

MLTE, the MultiLingual Text Engine, is available on Mac OS X. MLTE is a replacement for TextEdit and is a full Unicode text engine that uses ATSUI (Apple Type Services for Unicode Imaging). When an MLTE text object is created in an application, Unicode layout is included as well as support for 2-byte scripts. No need to install the necessary TSM handlers to support Japanese, Chinese, and Korean as MLTE does this. No need to install scrollbars, drag-and-drop handlers as MLTE can provide these services as well.

MLTE is an important piece of text rendering in Mac OS X. MLTE supports the usual QuickDraw anti-aliased text but it can also render on Mac OS X with Quartz. The result gives text-editing fields the look and feel typical to Mac OS X.

You'll find more information on MLTE at:
http://developer.apple.com/techpubs/macosx/Carbon/text/MultilingualTextEngine/Multilingual_Text_Engine/index.html

You'll find sample code on MLTE in the CarbonLib SDK, downloadable from:
http://connect.apple.com

What's Next?

If you need resources, or help localizing an application, visit: ***http://developer.apple.com/intl/*** for information. You'll find

Continued on page 43 ☛

"Built for Mac OS X" Artwork Now Available

Now that customers have Mac OS X in their hands, they'll be looking for great products to run on it. Tell the world that your product runs on Mac OS X by displaying the "Built for Mac OS X" badge on your product's packaging. The artwork, licensing requirements, and usage guidelines are available on the ADC Software Licensing web site.

http://developer.apple.com/mkt/swl/agreements.html#macosx

New Mac OS X Related Releases

Unless otherwise indicated, the following software is available from the Download Software area of the ADC Member Site at:
http://connect.apple.com/

• CarbonLib 1.4a2 SDK
The latest pre-release version of the CarbonLib 1.4 SDK for Mac OS, is now available to all ADC Members. This SDK provides all the files needed to begin Carbon development. CarbonLib 1.4 supports Mac OS 8.6 and greater.
http://connect.apple.com

• Mail Import Scripts 1.1
A bundle of AppleScripts that help you import mail messages from your current email program into Mail, the Mac OS X e-mail client.
http://asu.info.apple.com/swupdates.nsf/artnum/n12038/

• Mac OS X 10.0.3 Update
This update delivers CD-burning support for iTunes; a number of improvements for overall application stability; and latest version of the Internet file transfer service (ftpd), which features important security improvements.
http://asu.info.apple.com/swupdates.nsf/artnum/n12181/

• Cocoa Mailing List
The Cocoa development mailing list is a focal point for discussions on native Mac OS X application development using the Cocoa Frameworks: Foundation and Application Kit. Cocoa is based on advanced object oriented APIs that allow development in Java and Objective-C. Subscribers to this list will be discussing frameworks, features, and technical issues specific to Cocoa application development.
http://lists.apple.com/mailman/listinfo/cocoa-dev/

• Glossaries for Mac OS X
These are translated strings of commonly used words and phrases that developers can use in their applications. Download these updated glossaries in French, German, Italian, or Spanish.
http://developer.apple.com/intl/

Developer Documentation

The following new and updated documentation is available to help with successful Mac OS X application and peripheral development at:
http://developer.apple.com/techpubs/

Inside Mac OS X: Performance
Carbon Documentation
Aqua Human Interface Guidelines
Handling Carbon Events
Understanding Text Input and the Text Services Manager in Carbon
Unarchiving Interface Objects With Interface Builder Services
Interface Builder Services Reference
Text Services Manager Reference
Multilingual Text Engine Reference
Navigation Services Reference
Providing User Assistance With Apple Help
Apple Help Reference
Carbon Event Manager
Aqua Human Interface Guidelines
Carbon Documentation
Developing Cocoa Java Applications: A Tutorial
Aqua Human Interface Guidelines
Kernel Development
IO Kit Fundamentals
HID Device Interfaces
USB Device Interfaces

Technical Notes
TN2013 - The 'plst' Resource (Also available in Japanese)
TN2020 - Browser Plug-ins in Mac OS X
TN2002 - Compatibility between JDirect 2 and JDirect 3

Technical Q&As
QA1036 - Displaying PCI Configuration Resister contents in Open Firmware

Sample Code
SC - Cocoa: ToolbarSample
SC - Sound: PCI Sound Input Driver
SC - Interapplication Comm: BasicInputMethod
SC - Devices and Hardware: ATA: ATADemo

Finding Technical Information With Sherlock

Apple has created a set of Sherlock plug-ins for searching technical documentation on its Apple Developer Connection (ADC) web site. You can download these plug-ins from the web site at http://developer.apple.com/techpubs/indexes/sherlock/Sherlock_Files.sit. Install the plug-ins on a Mac OS X system by moving them to the Library/Internet Search Sites/Apple directory in your home directory. Install them on a Mac OS 9 system by dragging them to the Apple folder within the Internet Search Sites folder of your System Folder.

There is one plug-in for each set of technical documentation: Carbon, QuickTime, Mac OS X, Mac OS 9, hardware, and WebObjects. These plug-ins let you perform focused searches of the latest published documentation on Apple's web site in addition to using Help Viewer to search the documentation installed on Mac OS X.

Internationalization and Localization

Continued from page 41

resources for companies and organizations that can help you make the step to be international.

Remember, in order to be internationally correct in text rendering, you have to use Unicode. Unicode support in the system is achieved through MLTE and ATSUI. If you do Cocoa development you don't have to worry, all the text objects in Cocoa support Unicode layout and have built in support for Input Methods.

And last but not least, package your application! This will enhance the user experience of your customers and make your life easier.

Did You Know?

Guides for Internationalization

Apple's Technical Publications group is currently working on a book that will cover internationalization, localization, file encoding, and related issues for all Mac OS X application environments. However, you don't have to wait for that book to find out how to internationalize you application. Here are several sources of information:

• *Mac OS X: System Overview.* Includes a chapter on internationalization and multiscript support.
• *Project Builder Help.* Contains a section, under Files, on "Customizing for Different Regions and Platforms."
• *Cocoa Developer Documentation.* Includes a programming topic on internationalization for Cocoa applications.
• *Carbon Developer Documentation.* Includes several manuals on technologies that relate to internationalization and localization, including ATSUI, International Resources, and Text Encoding Conversion. These documents are listed under "Text and Other International Services" on the Carbon developer documentation home page.

You can access this documentation from the Help Viewer's Developer Help Center or from the ADC web site for Mac OS X documentation:
http://developer.apple.com/techpubs/macosx/macosx.html

Upcoming Seminars and Events

For more information on Apple developer events please visit the developer Events page at: ***http://developer.apple.com/events/***

Training and Seminars

R/com Offers Mac OS X Developer Training Online
R/com, also known as MediaSchool <www.mediaschool.com>, has partnered with Apple Developer Connection to create online training for Mac OS X developers. The first courses to be released in June include "Application Development for Mac OS X," "Carbon Development for Mac OS X," and "Cocoa: The Object-Oriented Application Solution." All classes have been reviewed by Apple engineers for technical accuracy. Check out their site to take a free virtual seminar, to learn more about current and upcoming courses, and to find out about the significant discounts offered to Premier, Select, and Student members of the Apple Developer Connection.
http://www.mediaschool.com/adc/

Apple iServices 5-day Cocoa Training
For application developers who want to learn how to develop Mac OS X applications using Cocoa, Apple iServices offers a five-day comprehensive, hand-on Cocoa training course. This course uses real-world examples and is perfect for developers who have a general understanding of Object-oriented concepts and practical experience with the C programming language or a C-derived language (Objective-C, Java, or C++). The course costs US $2,495.
http://www.apple.com/iServices/technicaltraining/cocoadev.html

Developer Related Conferences

1394 (FireWire) Developers' Conference 2000
June 31-Aug 2, Redmond, WA
Apple, Intel, and Microsoft are cosponsors
http://www.1394ta.org/Events/2001_DevCon/index.htm

MACWORLD Expo, New York 2001
July 17-20 in Jacob Javits Convention Center, NYC
Discounted exhibitor packages available
http://developer.apple.com/mkt/mwny2001.html

FileMaker Developer Conference 2001
August 12-15, Orlando, FL
More than 40 sessions and a product showcase
http://www.filemaker.com/devcon/
Various FileMaker training classes offered concurrently
http://www.DevconTraining.com

POWERPLANT WORKSHOP

By Aaron Montgomery

Basic Files

How one goes about writing a PowerPlant application

These Articles

This is the fourth article in a series of articles about Metrowerks' PowerPlant application framework. The first article introduced how the framework deals with commands, the second article discussed the debugging facilities of the framework and the third introduced windows. This article focuses on the file classes in the framework (opening and saving files). This series assumes familiarity with the C++ language, the Macintosh Toolbox API and the CodeWarrior IDE. The articles were written using CodeWarrior 6 with the net-update to IDE 4.1.0.3 and no other modifications. Throughout this and future articles, I will assume that you are using the class browser and the debugger to explore the PowerPlant code more fully.

Setting Up Navigation Services

Before you can use Navigation Services, you need to do some setup. The first step is to modify the common prefix file to let PowerPlant know whether you want to always use the classic file dialogs, require Navigation Services, or use Navigation Services if it is available and the classic dialogs otherwise. This is done in the project's prefix file.

CommonPrefix.h

```
#if PP_Target_Carbon
  #define PP_StdDialogs_Option \
    PP_StdDialogs_NavServicesOnly
  #define  SetTryNavServices_       do { } while (false)
#else
  #define PP_StdDialogs_Option  \
      PP_StdDialogs_Conditional
  #define SetTryNavServices_          \
    PowerPlant::UConditionalDialogs::SetTryNavServices(\
      0x01108000)
#endif
```

For Carbon targets, the code sets the dialogs to be Navigation Services at all times and the SetTryNavServices_ macro does nothing. For the other targets, the code uses conditional dialogs. The conditional dialogs option will use Navigation Services if it is available and classic dialogs otherwise. The SetTryNavServices_ macro tells PowerPlant to use Navigation services only if it has version at least 1.1 (PowerPlant stationery explains that earlier versions can cause problems).

The macro PP_StdDialogs_Option affects the UStandardDialogs.h header by setting the PowerPlant::StandardDialogs namespace (abbreviated to PP_StandardDialogs in the code) to equal one of PowerPlant::UNavServicesDialogs, PowerPlant::UConditionalDialogs, or PowerPlant::UClassicDialogs. When using the class browser to identify classes and functions, you will often find that the same class or function will appear in each of these three namespaces and you will need to look at the appropriate one.

The other necessary change to the code for Navigation Services is the need to load them at application startup and to unload them at application shutdown. This is done in CDocumentApp's constructor and destructor. In the constructor, the code uses the macro SetTryNavServices_ to establish what version it requires and then calls the function PP_StandardDialogs::Load(). In the destructor, the code calls PP_StandardDialogs::Unload().

As a final touch, I have also added BNDL, open, and kind resources to AppResources.rsrc. This provides us with custom icons and the finder with information about what types of files we can open.

Dirty Documents

Before discussing the methods used to open and save files, we will discuss how PowerPlant determines if a file has been modified (in other words, if the document is dirty). The LDocument class has an IsModified() method which should return true if the file has been modified since last read from disk and false otherwise. This allows the framework to enable and disable

Aaron teaches in the Mathematics Department at Central Washington University in Ellensburg, WA. Outside of his job, he spends time riding his mountain bike, watching movies and entertaining his wife and two sons. You can email him at montgoaa@cwu.edu, try to catch his attention in the newsgroup comp.sys.mac.oop.powerplant or visit his web site at mac69108.math.cwu.edu:8080/.

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

MACUSER EDITORS' CHOICE AWARD 1993 WINNER

Accelerated for Power Macintosh

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater, 1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red, International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

the Save and Revert menu commands appropriately. In order to make the system work, we will need to use the SetModified() functions when the file has been saved to the disk and when it has been modified by the user.

The first change to the code adds an iAmModified data member to the CHTMLTextView class. The LTextEditView class from which it derives provides the virtual method UserChangedText() which will be called anytime the user types (or deletes) something from the LTextEditView.

UserChangedText() in CHTMLTextView.cp

```
void CHTMLTextView::UserChangedText()
{
  if (IsModified() == false)
  {
    SetUpdateCommandStatus(true);
    SetModified(true);
  }
}
```

The code only needs to do something if this is the first modification. In that case the code tells the menus that they will need to be updated and then sets the object's flag to indicate that it has been modified. If you directly manipulate the text in an LTextEditView, the UserChangedText() method will not be called automatically. In our case, the InsertTag() method needs to indicate that it has modified the document and it does this with a call to UserChangedText(). The IsModified() and SetModified() methods of CHTMLView are inline functions which access and set the object's flag.

In the CTextDocument class, we change the IsModified() method so that it checks its CHTMLTextView's IsModified() method before returning a result. In addition, we update the SetModified() method so that it also calls the CHTMLTextView's SetModified() method. Note that PowerPlant does some housekeeping in LDocument's SetModified() method and so we call it from within the new method definition. You could also just copy the code, but that would mean that you would need to update your code if more housekeeping was done within LDocument's method.

Opening Files

We are now ready to consider how PowerPlant opens and saves files. The code presented here is modeled on the code in the stationery files provided with PowerPlant. There is one limitation with the strategy employed there that may make it unsuitable for your application. The problem arises because of the need to call ::NavCompleteSave() if Navigation Services have been used to save the file. The code in the PowerPlant stationery handles this by holding the file open on the disk until the document is closed. This allows the CTextDocument object to save the information necessary for the call to ::NavCompleteSave() until the document is closed. The disadvantage is that the file cannot be manipulated by another application while it is held open by HTMLedit. If you need to allow external tools to modify files while your application holds them open, you will need to adjust the code to permit this.

The first change in the code is the removal of some lines that were added in the first article. Now that our application can open files, the lines in FindCommandStatus() which disabled the open command can be removed. There is no need to change ObeyCommand() since PowerPlant's LDocApplication class already has the necessary code. Two CDocumentApp methods are necessary for opening files. The first, ChooseDocument(), interacts with the user to determine which file should be opened and the second, OpenDocument(), opens the document. We start with ChooseDocument().

ChooseDocument in CDocumentApp.cp

```
void
CDocumentApp::ChooseDocument()
{

  LFileChooser  theChooser;

  NavDialogOptions* theOptions =
                       theChooser.GetDialogOptions();
  if (theOptions != 0)
  {
  theOptions->dialogOptionFlags |= kNavSelectAllReadableItem;
  }

  if (theChooser.AskOpenFile(LFileTypeList()))
  {
    AEDescList    theDocList;
    theChooser.GetFileDescList(theDocList);
    SendAEOpenDocList(theDocList);
  }
}
```

There are really three LFileChooser classes, one in each of the namespaces mentioned earlier. At the top of the source file, the code indicates that the one in PowerPlant::StandardDialogs should be used. Therefore, the actual class used will be determined by the macros in the prefix file. All three of the LFileChooser classes allow you to obtain a pointer to a NavDialogOptions structure. The pointer will be 0 if the application is using classic dialogs. If the application is running under Navigation Services, we adjust the flags so that All Readable Items is the default choice from the popup menu in the dialog.

Next, the code creates an LFileTypeList (the default constructor uses the open resource with ID 128). In the case of HTMLedit, only TEXT files can be opened. If AskOpenFile() returns true, then the user requested a file be opened, and the code gets the AEDescList that describes the file from theChooser and sends an Open Apple Event to the application. PowerPlant will convert this Apple Event into a call to OpenDocument(). One nice feature is that there was almost no need to write separate code for the three possible situations (Navigation, Conditional, and Classic). Furthermore, you obtain the ability to open multiple files under Navigation Services without doing any extra work: PowerPlant will turn these lists into a sequence of individual open commands. We now turn our attention to the place where the document is actually opened.

OpenDocument() in CDocumentApp.cp

```
void
CDocumentApp::OpenDocument(
  FSSpec*     inMacFSSpec)
```

PAULIE'S
P·I·Z·Z·A·R·I·A
PAULIE'S

Domino's Pizza®

Marketing Rule no.1

MAKE YOUR PRODUCT EASIER TO BUY, YOU'LL SELL MORE OF IT.

You may have developed the best software in the world, but if nobody knows about it, or can find it, you won't sell much of it. With eSellerate, your application literally sells itself. Providing your customers with the power of instant gratification. And providing you with an opportunity of nearly limitless potential. **www.esellerate.net**

esellerate
The new way to sell software.

From MindVision Software, creator of VISE

```
{
  LDocument*  theDoc =
              LDocument::FindByFileSpec(*inMacFSSpec);

  if (theDoc != 0)
  {
    ValidateObject_(theDoc);
    theDoc->MakeCurrent();
  }
  else
    try
    {
      theDoc = NEW CTextDocument(this, inMacFSSpec);
      ValidateObject_(theDoc);
    }
    catch(LException& theErr)
    {
      if (theErr.GetErrorCode() != noErr)
      {
        throw;
      }
    }
}
```

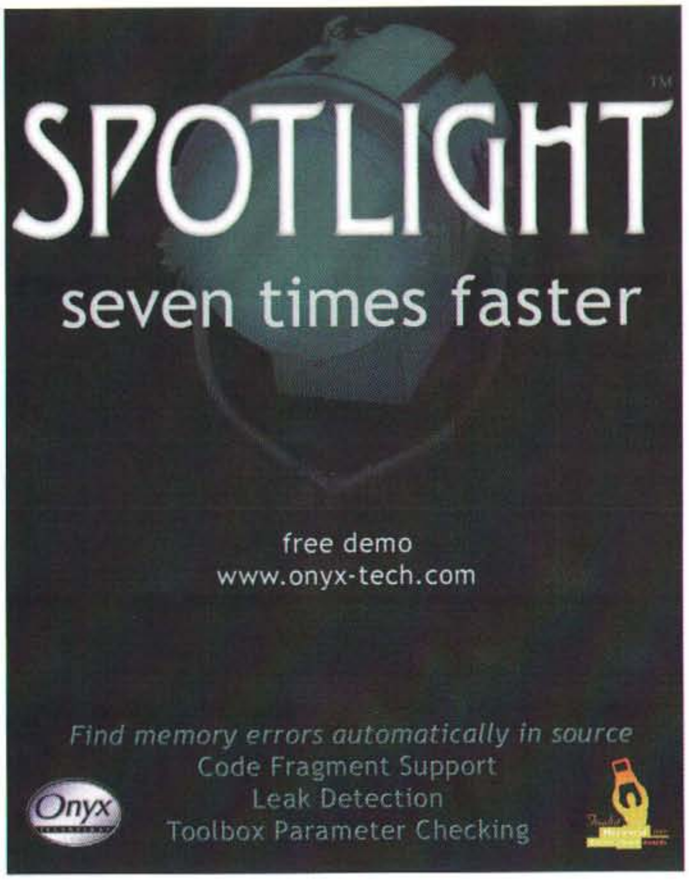
First, the code tries to find the document from among the currently open documents using the static **LDocument** method **FindByFileSpec()**. If **FindByFileSpec()** returns a pointer to an **LDocument**, then the code brings it to the front and is finished. If the document is not already open, then the code needs to create a new document and other than the error handling, this is accomplished to **CTextDocument**'s constructor.

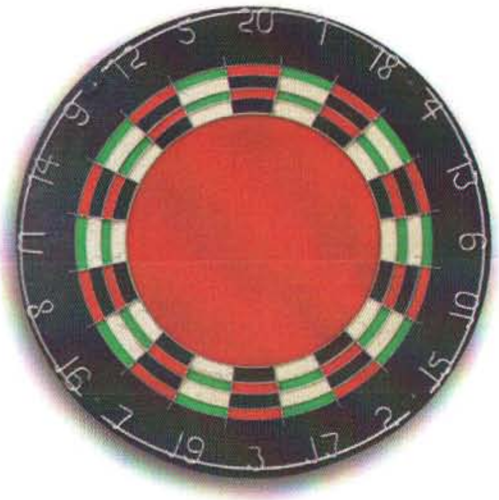
The error handling code deserves some mention here. The constructor of **CTextDocument** may throw an exception if the document contains more than 32K of text. While this problem is handled in the **OpenFile()** method of **CTextDocument** (presented below), an exception is the only way to inform the **CDocumentApp** that the file was not opened. My personal convention is to use an **LException** whose error code is **noErr** to indicate that everything has been handled but that the document is invalid. When the exception is thrown, DebugNew will report that the document leaked. If this were a "real" application, I would spend some time determining whether this is a real leak or if DebugNew cannot handle exceptions thrown from constructors. Since this is supposed to be a teaching article, I will leave this task as an exercise for the reader (watch out, Metrowerks pools its memory allocations).

You have now seen all of the significant changes to the **CDocumentApp** class and we turn our attention to the **CTextDocument** class. The **CTextDocument** constructor is modified slightly to accept an optional **FSSpec**. However almost all of the work is passed on to its **OpenFile()** method which is where we will pick up the trail. Just as **LSingleDoc** contains a pointer to an **LWindow** as one of its data members, it also contains a pointer to an **LFile** as another member. This is the real purpose of the **LDocument** class: to tie together the visual presentation (**LWindow**) with the data on the disk (**LFile**). The **LSingleDoc** class assumes that each document uses only one window and one file. Unlike **LWindow** which can be complicated (due to the visual hierarchy), the **LFile** class is simple. It will take care of handling the file reference numbers for both the data and resource forks and most of the methods of the class do exactly what their names indicate.

OpenFile() in CTextDocument.cp

```
void CTextDocument::OpenFile(FSSpec& inFileSpec)
{

  mFile = nil;

  try
  {
    StDeleter<LFile>theFile(NEW LFile(inFileSpec));
    ValidateObject_(theFile.Get());

    theFile->OpenDataFork(fsRdWrPerm);
    StHandleBlock theTextH(theFile->ReadDataFork());
    ValidateHandle_(theTextH.Get());
    ValidateObject_(myTextView);
    myTextView->SetTextHandle(theTextH);
    myTextView->SetModified(false);

    ValidateObject_(mWindow);
    mWindow->SetDescriptor(inFileSpec.name);

    mIsSpecified = true;

    mFile = theFile.Release();
  }
  catch (LException& theErr)
  {
    if (theErr.GetErrorCode() == err_32kLimit)
    {
      UModalAlerts::StopAlert(ALRT_BigFile);
      throw LException(noErr);
    }
    else
    {
      throw;
    }
  }
}
```





YES, IT IS THIS EASY.

At REAL Software, we like it simple. Take our award-winning product, REALbasic, for example. People call it the powerful, easy-to-use tool for creating their own software for Macintosh, Mac OS X and Windows. We call it a problem solver. You've probably said, "Wouldn't it be great if there was a little application that...." REALbasic fills that blank.

It's powerful and easy to use. Beginners and professionals alike can build software using a single, simple design. REALbasic compiles native applications for Macintosh, Mac OS X and Windows without requiring any platform-specific adjustments. Each version of your software looks and works just as it should in each environment.

Experiment, explore, learn and innovate as you create anything from prototypes to complete professional quality applications step by step. Simply drag and drop interface elements while REALbasic handles the details. You concentrate on what makes your stuff great — your ideas!

Complex problems shouldn't require complex solutions. The answer is REALbasic.

REALbasic3

Download a free demo. **www.realbasic.com**

The first goal of OpenFile() is to provide an LFile for the CTextDocument object. The code starts by creating a pointer to an LFile. The StDeleter class is analogous to the standard library's auto_ptr template and it is used here so that we do not leak the LFile pointer if an exception is thrown. The call to the StDeleter's Get() method returns the actual pointer and the code verifies that it is valid.

The code then opens the data fork of the LFile, copies the text into a handle (the StHandleBlock guarantees the memory is deleted at the end of the scope), and passes the handle to the document's CHTMLTextView object. Since the data in the CHTMLTextView is fresh from the disk, the code indicates that the CHTMLView is unmodified.

The code then sets the title of the window to match the name of the file. The mIsSpecified data member of the LDocument class indicates that this document has a file associated with it (important since it determines if the Revert command is valid). If everything went well, the code sets mFile. The call to Release() causes the StDeleter to disown the LFile pointer, failing to do this will cause DebugNew to (correctly) report a double-delete (once when the StDeleter goes out of scope and later in the CTextDocument's destructor).

The call to SetTextHandle() will throw an exception if the handle contains more than 32k characters. Since we know how to handle this problem here, we do so. The UModalAlerts method simply displays an alert stating that the file was too big (you need to supply the resource and PowerPlant does the rest). Since the code needs to let the calling function know that the file was not opened and there is no return value, I have opted to throw an LException with error code noErr. By personal convention, this means that something bad has happened but that no further remedies are needed. The first place where this error can be ignored should catch the exception and ignore it (you saw this in the OpenDocument() method of CDocumentApp). That completes the tour of the code necessary to open files using PowerPlant.

Although reversion of a document to the data on the disk is not essential, it is easy to add this ability to a PowerPlant application. PowerPlant stationery assumes that you will do this since it provides the Revert command in its File menu and adding the code usually adds little work.

DoRevert() in CTextDocument.cp

```
void CTextDocument::DoRevert()
{
  ValidateObject_(mFile);
  StHandleBlock theTextH(mFile->ReadDataFork());
  ValidateHandle_(theTextH.Get());

  ValidateObject_(myTextView);
  myTextView->SetTextHandle(theTextH);

  SetModified(false);

  myTextView->Refresh();
}
```

The code reads the data from the CTextDocument's mFile into a handle (using a StHandleBlock to prevent a leak). Then it updates the text in the CHTMLTextView. The code then indicates that the document is now clean. The last step is to make a call to Refresh(). One important thing to be careful with here is that the call to Refresh() resolves into a call to ::InvalPortRect(). This means that no actual drawing will be done until the next Update event is handled. Therefore, if you have stopped the event processing queue, then calls to Refresh() will appear to do nothing.

Although I will not discuss it in this article, I have also made the NameNewDocument() method a little more sophisticated. In retrospect, this really should have been done in the third article where we discussed windows. You might want to examine the code on your own. Now we turn to saving (which is a little more complicated).

Saving Files

All saving is handled by the CTextDocument class (and not by the CDocumentApp class). Three methods need to be implemented for PowerPlant to handle saving files. The first method, AskSaveAs(), presents the user with a dialog to determine where to save the file. The second method, DoAESave(), method implements as Save As operation (which handles the Save Apple Event). The third method, DoSave(), is the method that actually writes the data to the disk.

The LDocument class has an implementation for AskSaveAs(). Unfortunately, it has not been updated to handle the need for a ::NavCompleteSave() call under Navigation Services. In order to handle this, we need to rewrite the AskSaveAs() command as well as retain a new data member: myFileDesignator.

AskSaveAs() in CTextDocument.cp

```
Boolean CTextDocument::AskSaveAs(
  FSSpec& outFSSpec, Boolean inRecordIt)
{

  Boolean     didSave = false;

  StDeleter<LFileDesignator>
    theDesignator(NEW LFileDesignator);
  ValidateSimpleObject_(theDesignator.Get());

  theDesignator->SetFileType(GetFileType());
  NavDialogOptions* theOptions =
    theDesignator->GetDialogOptions();
  if (theOptions != 0)
  {
    theOptions->dialogOptionFlags |= kNavNoTypePopup;
  }

  Str255 theDefaultName;
  if (theDesignator->AskDesignateFile(
    GetDescriptor(theDefaultName)))
  {

    theDesignator->GetFileSpec(outFSSpec);
    if (UsesFileSpec(outFSSpec))
    {

      if (inRecordIt)
      {
        SendSelfAE(kAECoreSuite, kAESave, ExecuteAE_No);
      }

      DisposeOf_(myFileDesignator);
      myFileDesignator = theDesignator.Release();

      DoSave();
```

Sometimes, one just isn't enough...

DVIator for ADC

... and sometimes it is.

With DVIator and a DVI video card you can add additional ADC displays to a new G4, or upgrade an older Mac with Apple's new flat-panels.

Dr. Bott LLC

www.drbott.com • 877.611.2688 • 503.582.9944

```
      didSave = true;

    }
    else
    {

      if (inRecordIt)
      {
        SendAESaveAs(outFSSpec, GetFileType(),
          ExecuteAE_No);
      }

      if (theDesignator->IsReplacing())
      {
        ThrowIfOSErr_(::FSpDelete(&outFSSpec));
      }

      if (myFileDesignator != 0)
      {
        ValidateSimpleObject_(myFileDesignator);
        myFileDesignator->CompleteSave();
        DisposeOf_(myFileDesignator);
      }

      myFileDesignator = theDesignator.Release();

      DoAESave(outFSSpec, fileType_Default);

      didSave = true;
    }
  }

  return didSave;
}
```

The first thing the code does is to create a new LFileDesignator (again, a StDeleter is used to prevent leaks). Before interacting with the user, we need to adjust some settings for the dialog. Just as with the LFileChooser in ChooseDocument() above, the GetDialogOptions() will return 0 if it is running under classic dialogs. We adjust the options so that the user will not be presented with a type popup menu. This is consistent with the fact that we only save TEXT documents from this application. The code also sets the default name for the file to the name of the window.

The call to AskDesignateFile() returns true if the user has decided to save the file (and false if they cancel the dialog). All the information obtained from the dialog is available through theDesignator. The first thing we do is set outFSSpec to the FSSpec obtained from interaction with the user. Then we determine if the document's existing file is being overwritten or the document is writing to another file. The two possibilities are similar and we discuss them in parallel.

In both cases the first thing the code does is to send the application an Apple Event if the application's Apple Events are being recorded. The actual Apple Event sent is different, but the code is very similar. The parameter ExecuteAE_No indicates that the Apple Event should not be executed once it is received. This is appropriate since the command is being handled here and there is no need for the code to save the file twice.

If the user is saving to a different file and that file already exists, then the file needs to be deleted and this is done next (in the second branch). Now it is time to handle the changes to the myFileDesignator data member. If we are overwriting our original file on disk, then the old LFileDesignator is deleted and the new designator is saved. Since the file is not being closed, the code does not need to call CompleteSave(). In the other case (where the file on the disk is changing and the LFileDesignator is not 0) the code calls CompleteSave() before disposing of the old LFileDesignator. Next either DoSave() or DoAESave() is called. In both of these cases, the function will then return true because the file has been saved.

Next we examine the DoAESave() method. This method implements a Save As command and calls DoSave() to do the actual work of writing to the disk. Almost all of the code is generic and you should be able to use it "out of the box." In fact, only the OSType_Creator constant is application specific and could easily be factored into a call to a new virtual function with the name GetCreatorType().

DoAESave() in CTextDocument.cp

```
void CTextDocument::DoAESave(
  FSSpec& inFileSpec, OSType inFileType)
{
  DisposeOf_(mFile);
  mIsSpecified = false;

  StDeleter<LFile>theFile(NEW LFile(inFileSpec));
  ValidateObject_(theFile.Get());

  OSType theFileType = GetFileType();
  if (inFileType != fileType_Default)
  {
    theFileType = inFileType;
  }

  theFile->CreateNewFile(OSType_Creator, theFileType);
  theFile->OpenDataFork(fsRdWrPerm);
  theFile->OpenResourceFork(fsRdWrPerm);

  mFile = theFile.Release();

  DoSave();

  ValidateObject_(mWindow);
  mWindow->SetDescriptor(inFileSpec.name);

  mIsSpecified = true;
}
```

Since this is a Save As operation, the code first eliminates the existing LFile object. This does not delete the file on the disk, but it will close the file if necessary. Notice that this is done even before we know that the new file is valid. My reasoning is that after an attempted Save As command, the goal is to protect the original file's data (and prevent a Save command from clobbering it). The DisposeOf_ macro will set the mFile data member to 0 and setting mIsSpecified to false will prompt the user with another AskSaveAs() dialog if they try to close the window after a failed Save As command.

Next, the code creates the file on the disk (with another StDeleter). Once we know the file has been successfully created and opened, the CTextDocument object will take control of the LFile pointer (again the call to Release() is important to avoid a double-delete). The DoSave() command expects that both the resource and data forks of the file are open for writing. Once DoSave() writes the data to the disk, the document's window title is updated and its mIsSpecified data member is set to true. We now turn our attention to the one method that requires knowledge of the data layout in the files: DoSave().

DoSave() in CTextDocument.cp

```
void CTextDocument::DoSave()
{
  ValidateObject_(myTextView);
  Handle theTextH = myTextView->GetTextHandle();
  ValidateHandle_(theTextH);
  Size theTextSize = ::GetHandleSize(theTextH);

  StHandleLocker  theLock(theTextH);

  ValidateObject_(mFile);
  mFile->WriteDataFork(*theTextH, theTextSize);

  SetModified(false);
}
```

Saving text files is rather simple: get a handle to the text and write it out to the file's data fork. The last line indicates that the document currently matches the data on the disk. Somewhat anti-climatic, but it works. The actual PowerPlant stationery file saves some information in the resource fork as well, but I will leave that for you to explore on your own.

Concluding Remarks

Although Opening, Reverting and Saving files requires a significant amount of code, most of the work is done by PowerPlant. In fact, of the methods you need to implement, most of them can be lifted almost verbatim from the PowerPlant stationery files. At this point, I think I have covered the code presented in the PowerPlant Advanced stationery file (with a few omissions that you should be able to work out using the class browser and debugger). In the next article (the fifth of this series), I will spend some time talking about control classes which can be used to liven up your dialogs. Once that topic is covered, I feel I will have covered the essential core of PowerPlant. At this point, I will reiterate my request that people should indicate what topics they would like to see. Here is a short list of topics I have considered: more on menus, more on panes, threads, drag & drop, tables, actions (and undo strategies), Apple Events, contextual menus. I am flexible and willing to look at other topics if they are suggested.

PowerPlant References

I could not find as many references on files as on some of the other topics, there is a single chapter in **The PowerPlant Book** entitled "File I/O". You should also spend some time sifting through the source code of PowerPlant as well as the example files (there is a StdDialogs demo as well as a TextDocument demo). MT

5 times faster

New: PrimeBase 4.0 !

Using PrimeBase, you are able to produce Internet and Intranet Applications faster and with the best performance ever.

The development package includes the

- PrimeBase Application Server
- PrimeBase Development Framework
- PrimeBase SQL Database Server

Develop on a Mac and deploy without any additional effort on any platform you want.

- PrimeBase is cross-platform
- Full text index
- Object oriented java like language
- Separation of code and layout
- Native connectivity to any database

free developer version

download now !

▶ www.primebase.com

PrimeBase

Snap Innovation GmbH
Altonaer Postraße 9a
D-22767 Hamburg / Germany
www.primebase.com info@primebase.com
Fon:++49 (40) 30629-400
Fax:++49 (40) 30629-499

by Tim Monroe

Back In Action

Working with Wired Actions

INTRODUCTION

The previous four QuickTime Toolkit articles have focused largely on two topics: how to create and manipulate sprites, and how to attach dynamic, interactive behaviors to sprites. Interactivity is one of QuickTime's greatest assets and perhaps its most significant advantage over competing media architectures. And, no doubt, wired sprites are the cornerstone of QuickTime interactivity. As we've seen, we can do some pretty nifty things with just a few event and action atoms placed in the right locations in a sprite track.

In this article, we're going to take one more look at wired actions. I want to consider a few of the event types and actions recently introduced in QuickTime 5, and I especially want to consider how to use wired actions with other kinds of tracks. So far, all the actions we've encoountered have been triggered by some event involving a sprite (for example, clicking on the sprite) or a sprite track (for example, loading a sprite track key frame). Beginning in QuickTime 4, however, it's also possible to attach wired actions to QuickTime VR nodes and hot spots, to text in text tracks, and to Flash content. Let's take a moment to see what we can do here.

QuickTime VR supports two general kinds of wired actions: (1) actions that apply to a particular node, and (2) actions that apply to a particular hot spot in a node. The node-specific actions can be attached to idle events (so that an action can be triggered periodically) or to frame-loaded events (so that an action can be triggered when the node is first entered). We could, for instance, attach an action to a frame-loaded event to set the initial pan and tilt angles that are displayed when the user enters the node. Similarly, we could attach an action to a frame-loaded event to start a sound track playing when the user enters the node. In addition, we could use idle event actions to dynamically adjust the balance and volume of that sound as the user changes the pan angle of the movie; this provides an easy way to make a sound appear to emanate from a specific location in a node (which is sometimes called directional sound). Or, we could adjust the pan and tilt angles of some other QuickTime VR movie, so that panning and tilting one panorama caused similar panning and tilting in a second.

We can also attach wired actions to hot spots inside of QuickTime VR nodes. We can configure these actions to be triggered by a variety of mouse events, including moving the mouse over a hot spot, moving the mouse out of a hot spot, and clicking the mouse within a hot spot. Once again, the actions that are triggered by these events can be any actions supported by the QuickTime wired actions architecture, such as starting and stopping sounds, enabling and disabling tracks, changing pan and tilt angles, and so forth.

QuickTime allows us to attach wired actions to a text track, giving us wired text. Consider, for instance, the movie shown in **Figure 1**. This movie contains a text track that's wired to displays the amount of memory currently available in the application's heap. (The text track is updated every two seconds by an idle event action.) We can use this movie to track the memory usage of any QuickTime-savvy application.

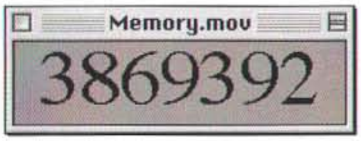


***Figure 1**: A text track displaying the application's free memory*

Or consider the movie shown in **Figure 2**, which also contains a single text track. This time the text track is wired so that clicking on the word "Apple" launches the user's default web browser and loads the URL <http://www.apple.com/>; similarly, clicking on the word "CNN" loads the URL <http://www.cnn.com/>.

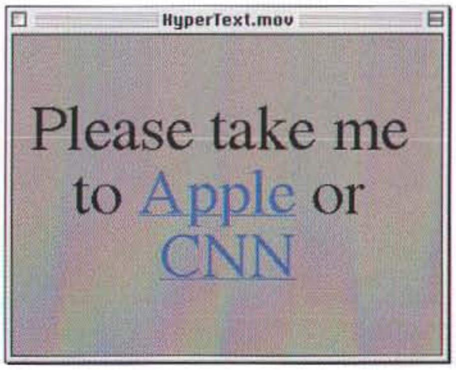


***Figure 2**: A text track with hypertext links*

Tim Monroe works in the QuickTime engineering team at Apple Computer, Inc. You can contact him at monroe@apple.com.

Software Developers

HASP4 for Mac - Gear Up for Increased Revenue

NOW SUPPORTS OS X!

HASP4 protects intellectual property and increases revenues by preventing unauthorized software usage and duplication. HASP4 features:

- A new proprietary design prevents reverse engineering
- A variety of solutions, including single users, network floating or concurrent licenses, or time-based subscriptions/trials
- Available in parallel, USB, ADB and PCMCIA versions
- Compatible with Windows 95/98/NT/2000; Mac OS 8/9/X and Linux

Order your HASP4 for Mac Developer's Kit at:

www.eAladdin.com/mactech

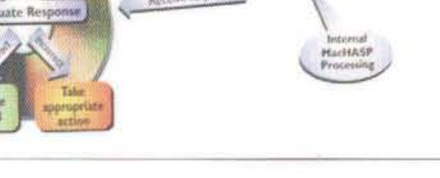
Or call 800-223-4277 EST or 800-562-2543 WST, CST & MST.

ALADDIN

Securing the Global Village

Identification Card

HASP4
The Professional Software Protection System

Name:	HASP4 for Mac
Main Mission:	To protect your software & increase your revenues
HASP is:	A small hardware key that connects to the computer and prevents unauthorized use of your software.
Features & benefits	Prevents software piracy Controls software distribution Protects intellectual property Increases revenues Reduces software distribution costs
Product Diagram:	

International HASP@eAladdin.com **North America** HASP.us@eAladdin.com **UK** HASP.uk@eAladdin.com **Germany** HASP.de@eAladdin.com
Benelux HASP.nl@eAladdin.com **France** HASP.fr@eAladdin.com **Israel** HASP.il@eAladdin.com **South Africa** HASP.za@eAladdin.com
Brazil HASP.br@eAladdin.com **Japan** HASP.jp@eAladdin.com **Russia** HASP.ru@eAladdin.com **Spain** HASP.es@eAladdin.com **China** HASP.cn@eAladdin.com

Technology for Life

Macworld Conference & Expo™

Conference Programs & Workshops
July 17-20, 2001

Exposition
July 18-20, 2001

Jacob K. Javits Convention Center
New York City

Register Online Today!
www.macworldexpo.com
Call Toll Free 1-800-645-EXPO

Register by June 18, 2001 to SAVE $150 on a Macworld Conference & Expo SUPER PASS!

Owned and Managed by IDG WORLD EXPO

Register Online at www.macworldexpo.com
with your Priority Code: A-MTC4

by June 18, 2001 to take advantage of Special Savings!

Our long-standing dedication to technology development merits Macworld Conference & Expo as the ultimate venue for all levels of Mac users, professionals, enthusiasts and the Mac-curious to gather and experience the excellence of Mac technology!

The most important Macworld Conference yet!

Workshops

Tuesday, July 17, 2001

The week begins with 13 astounding, full-day workshops that provide in-depth training on key products and technologies. You can find full workshop descriptions at www.macworldexpo.com.

Macworld/Pro Conference Program

Wednesday, July 18 – Friday, July 20, 2001

Macworld/Pro offers the most sophisticated training available on Macintosh for the advanced users and skilled professionals. Six distinctive tracks include:

- **Macintosh Networking and Communications**
- **Mac OS X in Depth**
- **Professional Publishing**
- **Application Spotlight: FileMaker Pro**
- **The Mac Manager Track**
- **Digital Media**

World-Class Exposition!

Wednesday, July 18 – Friday, July 20, 2001

Excel in life with the knowledge and solutions found at Macworld Conference & Expo — the world's most comprehensive Macintosh OS event!

- Visit over 400 exhibiting companies
- Discover thousands of new products and services
- Test-drive the latest Mac OS X applications
- Participate in live demonstrations
- Evaluate the latest technological innovations

Macworld/Users Conference Program

Wednesday, July 18 – Friday, July 20, 2001

The Macworld/Users Conference continues to be one of the best educational values anywhere, offering over 80 educational sessions on a variety of exciting topics presented by industry experts! Mac users and enthusiasts can learn about Mac OS X — your first taste!, Desktop Movies, Digital Photography, Tips about your Favorite Applications, Digital Imaging and much more!

Many Macworld/Users sessions can be combined as curriculums for

- Creative Professionals
- Small Business Owners
- Musicians
- Educators

After great success last summer, MacBeginnings returns to New York! Open to ALL registered attendees!

MacBeginnings

Wednesday, July 18 – Friday, July 20, 2001

Visit www.macworldexpo.com for session times and descriptions.

Brand New for New York!

The hottest up-and-coming companies and developers in the Mac industry will be on display as you stroll down the **Special Interest Boulevard** and visit **MacTech Central.**

Visit www.macworldexpo.com for the most up-to-date exhibitor list.

Flagship Sponsors

Macworld Macworld.com MacCentral.com

Finally, we can attach wired actions to items in a Flash track. *Flash* is a file format developed by Macromedia, Inc. for displaying vector-based graphics and animations. It's especially popular for web-based content delivery, since it combines compact data size (and hence speedier downloads) with quick rendering on the client, advanced graphics manipulations, and support for mouse-based interactivity with the graphics objects. In QuickTime 4, Apple added a Flash media handler that provides support for Flash graphics, animations, and interactivity inside of QuickTime movies. In addition, QuickTime 4 added the ability to attach wired actions to elements in a Flash track, thereby supplementing the native Flash interactivity. So, for instance, we could use Flash graphic elements (instead of sprites) to provide a user-interface for a QuickTime movie.

Unfortunately, we don't yet know enough about the structure of QuickTime VR movies or Flash tracks to know where to insert event atoms. As a result, we'll have to postpone our hands-on work with wiring VR or Flash content. But we *do* know how text tracks are put together (see "Word Is Out" in *MacTech*, November 2000), so here we'll learn how to add some interactivity to our text tracks.

Our sample application this month is called QTWiredActions. The Test menu of QTWiredActions is shown in **Figure 3**. The first menu item creates the wired text movie shown in **Figure 2**. The second menu item creates the memory display movie shown in **Figure 1**.

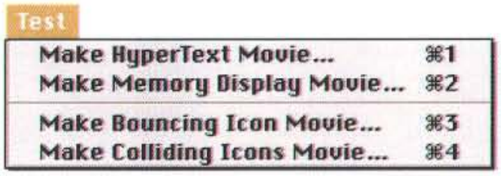

Test
Make HyperText Movie... ⌘1
Make Memory Display Movie... ⌘2
Make Bouncing Icon Movie... ⌘3
Make Colliding Icons Movie... ⌘4

***Figure 3**: The Test menu of QTWiredActions*

The next two menu items create two more sprite movies, which use wired actions and operands to make sprites react "physically" with the movie rectangle and with one another.

TEXT ACTIONS

Let's begin by investigating a few of the ways in which we can use wired actions in text tracks. In general, we attach an event atom to a text media sample by concatenating the atom container that holds the event atom onto the media sample data. (We'll see exactly how this works in a moment.) The event atom can specify any of the kinds of events that we've considered so far, including mouse-click events, mouse-over events, frame-loaded events, and idle events. These wired text samples work exactly as you'd expect; for instance, if a text sample contains a mouse-click event atom, then the associated actions are triggered when the user clicks the mouse button while the cursor is within the bounds of the text track.

Sometimes, however, we'd like to handle user events that involve only *part* of the text displayed in a text sample. Consider once again the text movie shown in **Figure 2**. In this case, we want to trigger actions only when the user clicks on specific portions of the text data (namely, the strings "Apple" and "CNN"). To handle this kind of wiring, QuickTime supports a class of atoms called *hypertext atoms*. The goal here is to provide the sort of hyperlinks that you typically find in web browsers or other HTML-based applications. By clicking on a hypertext link in a text track, the user can launch the default web browser and navigate to a specific URL, by triggering the kActionGoToURL action. But in fact hypertext atoms can contain any kinds of wired action, not just kActionGoToURL actions. So the user's actions can just as easily (for instance) trigger a jump forward or backward in the movie, or cause some changes in an external movie.

As you can see in **Figure 2**, QuickTime automatically sets the color of hypertext links to the familiar browser-default blue and underlines the hypertext. Both of these provide visual cues that the user can interact with that segment of text. QuickTime also provides wired actions that allow us to change the color of a segment of hypertext, for example when the user rolls the cursor over that segment or after the user has clicked on the link. **Figure 4** shows the same hypertext movie (this time on Windows) with the cursor resting over the first hyperlink.

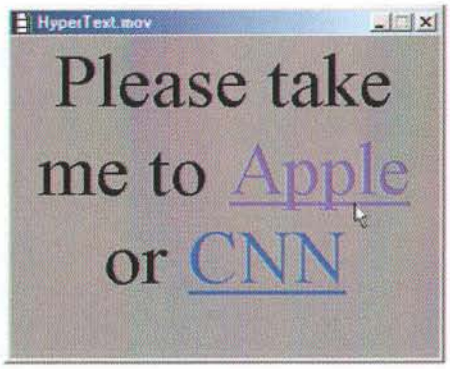
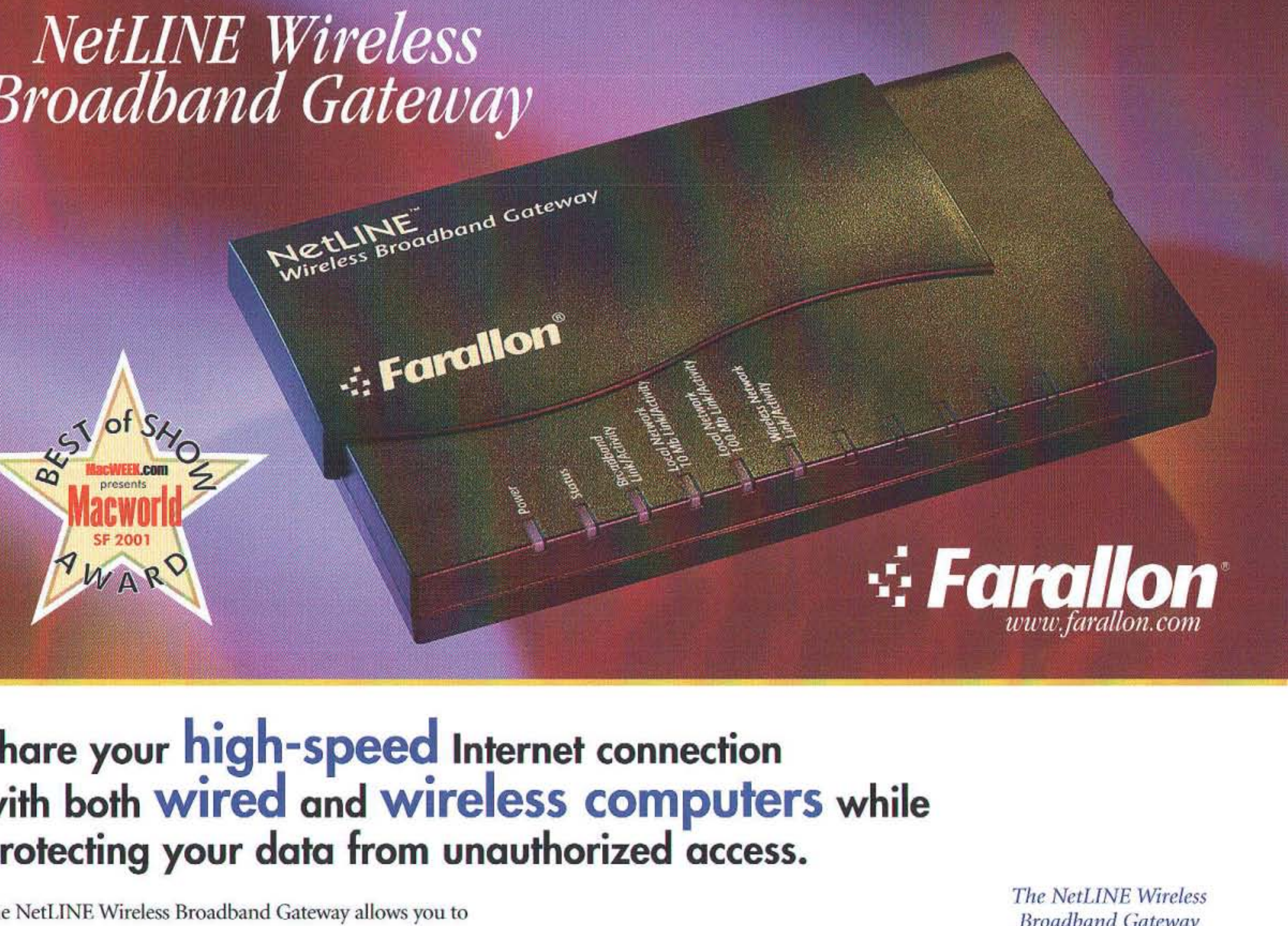


***Figure 4**: A selected hypertext link*

Adding Actions to a Text Sample

A text media sample consists of a 16-bit length word followed by the text of that sample. Optionally, one or more atoms of additional data (called *text atom extensions*) may follow the text in the sample. The length word specifies the total number of bytes in the text (not including the 2 bytes occupied by the length field itself or the length of any of the optional text atom extensions). The text atom extensions are organized as "classic" atom structures: a 32-bit length field, followed by a 32-bit type field, followed by the data in the atom. Here, the length field specifies the total length of the atom (that is, 8 plus the number of bytes in the data). All the data in a text extension atom must be in big-endian format.

Not Your Average Wireless Access Point



hare your high-speed Internet connection vith both wired and wireless computers while rotecting your data from unauthorized access.

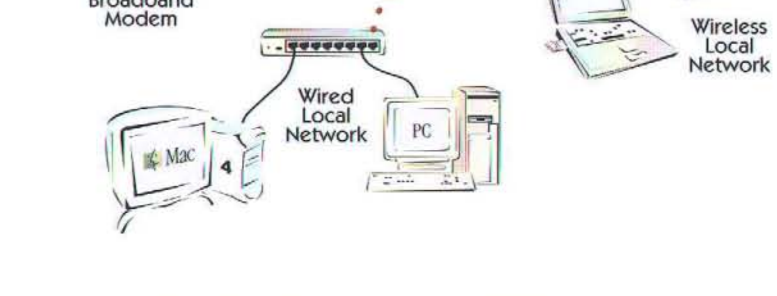
he NetLINE Wireless Broadband Gateway allows you to are a high-speed Internet connection, such as Cable or SL, with multiple computers using just a single IP ddress from your ISP. Support for both 802.11b wireless nd standard Ethernet connections are built-in to this router nd access point all in one.

he built-in firewall will protect your important files by reventing unauthorized access via the Internet and the eb-based configuration makes set-up a breeze.

00% compatible with SkyLINE, Airport, nd all other 802.11b wireless cards.

or more information contact Dr. Farallon t 1-800-613-4954 or visit us on the web at **ww.farallon.com.**

Available @ The DEPOT



The NetLINE Wireless Broadband Gateway connects between your network and your Cable/DSL modem

Whether we use a hypertext atom or any of the standard types of event atoms, we add the event atom to a text media sample in the same way, by attaching the atom container that contains the event and action atoms to the text media sample as a text atom extension. **Figure 5** shows the general structure of a text media sample that contains a text action.

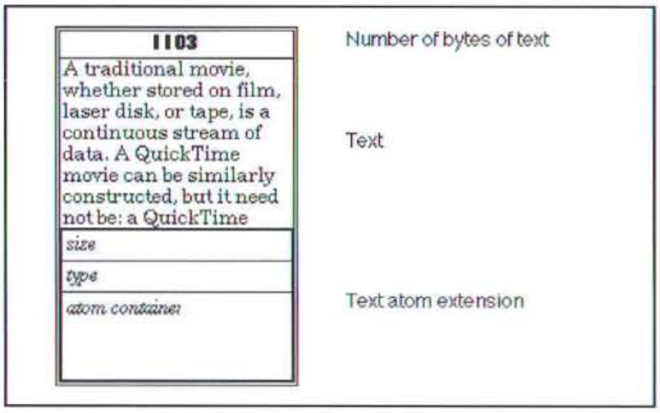

1103
A traditional movie, whether stored on film, laser disk, or tape, is a continuous stream of data. A QuickTime movie can be similarly constructed, but it need not be: a QuickTime
size
type
atom container
Number of bytes of text
Text
Text atom extension

***Figure 5**: The structure of a wired text media sample*

If we're given a text media sample and an event atom container, it's quite easy to wire that atom container to the text. We simply need to determine the lengths of the media sample and the atom container, enlarge the text media sample to hold all of its existing data and the atom container, plus the 8-byte atom header shown in Figure 5. We set up the atom header as appropriate and then copy it and the atom container into the enlarged media sample.

The size of a text atom extension for wired actions, of course, is the size of the atom container plus the size of the atom header (8 bytes). The type of the text atom extension can be one of three values. If the atom container holds a frame-loaded event atom, then the type of the text atom extension should be set to kQTEventFrameLoaded. If the atom container holds any other standard event atom (including an idle event atom), then the type of the text atom extension should be set to kQTEventType. Finally, if the atom container holds a hypertext event atom, then the type of the text atom extension should be set to 'htxt'. Currently there is no symbolic constant for this value defined in any of the public QuickTime header files, so I've included this definition in the file QTWiredActions.h:

```
#define kHyperTextTextAtomType     FOUR_CHAR_CODE('htxt')
```

Listing 1 shows our definition of the function QTWired_AddActionsToSample, which adds an event atom container to an existing text media sample. Notice that the function parameters include the text media sample, the event atom container, and the type of atom extension. This type should be one of the three recognized types for text atom extensions that specify wired actions.

Listing 1: Adding wired actions to a text media sample

QTWired_AddActionsToSample

```
static OSErr QTWired_AddActionsToSample (Handle theSample,
        QTAtomContainer theActions, SInt32 theAtomExtType)
{
  Ptr       myPtr = NULL;
  long      myHandleLength;
  long      myContainerLength;
  long      myNewLength;
  OSErr     myErr = noErr;

  if ((theSample == NULL) || (theActions == NULL))
    return(paramErr);

  myHandleLength = GetHandleSize(theSample);
  myContainerLength = GetHandleSize((Handle)theActions);

  myNewLength = (long)(sizeof(long) + sizeof(OSType) +
                      myContainerLength);

  SetHandleSize(theSample, (myHandleLength + myNewLength));
  myErr = MemError();
  if (myErr != noErr)
    goto bail;

  HLock(theSample);

  // get a pointer to the beginning of the new block of space added to the sample
  // by the previous call to SetHandleSize; we need to format that space as a text
  // atom extension
  myPtr = *theSample + myHandleLength;

  // set the length of the text atom extension
  *(long *)myPtr = EndianS32_NtoB((long)(sizeof(long) +
                      sizeof(OSType) + myContainerLength));
  myPtr += (sizeof(long));

  // set the type of the text atom extension
  *(OSType *)myPtr = EndianS32_NtoB(theAtomExtType);
  myPtr += (sizeof(OSType));

  // set the data of the text atom extension;
  // we assume that this data is already in big-endian format
  HLock((Handle)theActions);
  BlockMove(*theActions, myPtr, myContainerLength);

  HUnlock((Handle)theActions);
  HUnlock(theSample);

bail:
  return(myErr);
}
```

For more information on working with atoms, see "The Atomic Café" in *MacTech*, September 2000.

Creating Text Actions

As we've just seen, we add a wired action event handler to a text sample by adding a text atom extension of type kQTEventFrameLoaded or kQTEventType to the end of the sample; the data in the text atom extension is the atom container that holds the information about the wired actions triggered by some event. So, our task boils down to this: find the data in a text sample, create an atom container holding information about the desired actions, and then append a text extension atom whose data is that atom container to the end of the text sample data. Then we replace the previous text sample with the new one in the text track.

Listing 2 shows the code that handles the "Make Memory Display Movie..." menu item.

Listing 2: Handling the "Make Memory Display Movie..." menu item

QTApp_HandleMenu

```
case IDM_MAKE_MEM_DISPLAY_MOVIE:
  myPrompt = QTUtils_ConvertCToPascalString(kMDSavePrompt);
  myName = QTUtils_ConvertCToPascalString(kMDSaveFileName);

  // elicit a file from the user to save the new movie into
  QTFrame_PutFile(myPrompt, myName, &myFile, &myIsSelected,
                         &myIsReplacing);

  // create a text movie that displays the amount of memory free in the application
heap
  if (myIsSelected) {
    myErr = QTWired_CreateMemoryDisplayMovie(&myFile);
    if (myErr == noErr)
      QTWired_AddActionsToTextMovie(&myFile, theMenuItem);
  }

  myIsHandled = true;
  break;
```

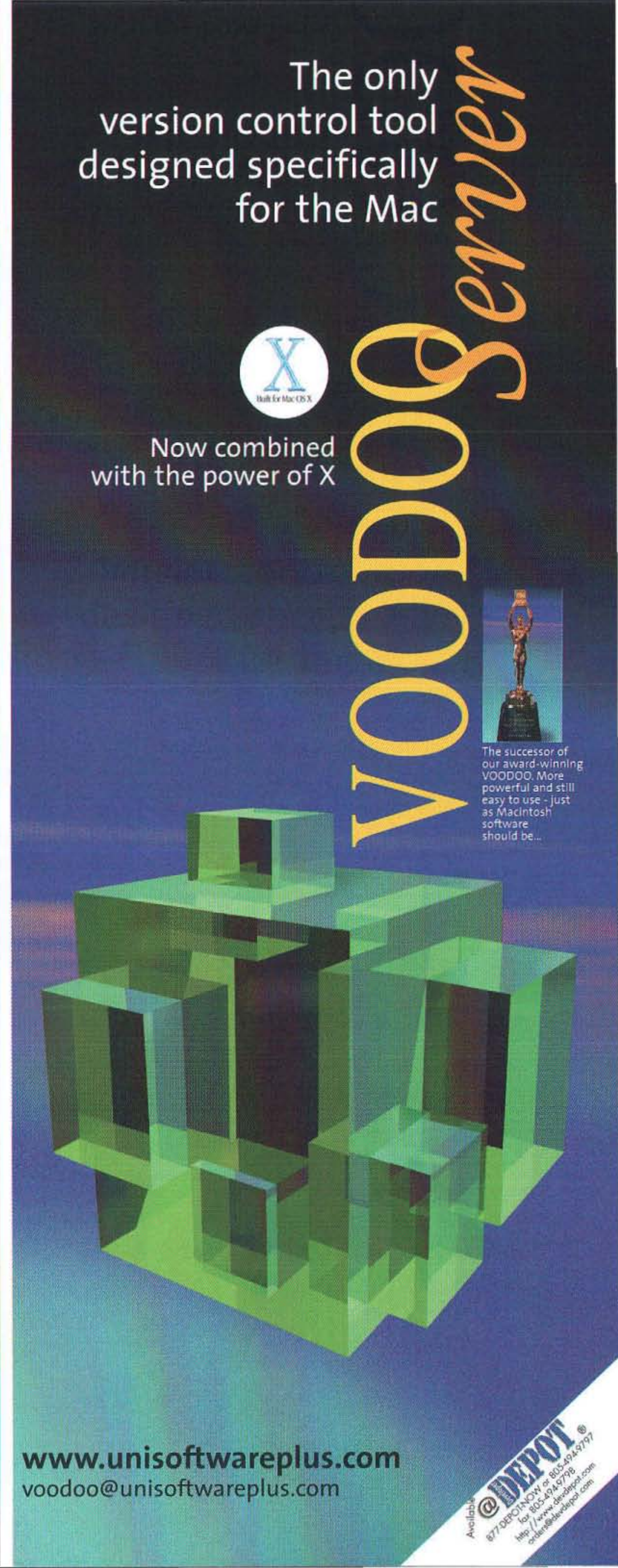
We don't need to consider the function QTWired_CreateMemoryDisplayMovie in detail, since it's really just a variant of the QTText_AddTextTrack function we considered in an earlier article. In the present case, we set the initial text in the text track to the single character "0", and we set the font to 48-point Times-Roman (using the constant kFontIDTimes).

The interesting work in Listing 2 is done by the QTWired_AddActionsToTextMovie function, which creates the appropriate wired action atom container and then calls QTWired_AddActionsToSample (defined in Listing 1) to append that container to the first (and only) text media sample in the file created by QTWired_CreateMemoryDisplayMovie. Listing 3 shows our definition of QTWired_AddActionsToTextMovie.

Listing 3: Adding wired actions to a text movie

QTWired_AddActionsToTextMovie

```
OSErr QTWired_AddActionsToTextMovie
                  (FSSpec *theFSSpec, UInt16 theMenuItem)
{
  short                    myResID = 0;
  short                    myResRefNum = -1;
  Movie                    myMovie = NULL;
  Track                    myTrack = NULL;
  Media                    myMedia = NULL;
  TimeValue               myTrackOffset;
  TimeValue               myMediaTime;
  TimeValue               mySampleDuration;
  TimeValue               mySelectionDuration;
  TimeValue               myNewMediaTime;
  TextDescriptionHandle    myTextDesc = NULL;
  Handle                   mySample = NULL;
  short                    mySampleFlags;
  Fixed                    myTrackEditRate;
  QTAtomContainer          myActions = NULL;
  OSErr                    myErr = noErr;

  // open the movie file for reading and writing
  myErr = OpenMovieFile(theFSSpec, &myResRefNum, fsRdWrPerm);
  if (myErr != noErr)
    goto bail;

  myErr = NewMovieFromFile(&myMovie, myResRefNum, &myResID,
                NULL, newMovieActive, NULL);
  if (myErr != noErr)
    goto bail;

  // find first text track in the movie
  myTrack = GetMovieIndTrackType(myMovie, kIndexOne,
                TextMediaType, movieTrackMediaType);
  if (myTrack == NULL)
    goto bail;

  // get first media sample in the text track
  myMedia = GetTrackMedia(myTrack);
  if (myMedia == NULL)
    goto bail;
```



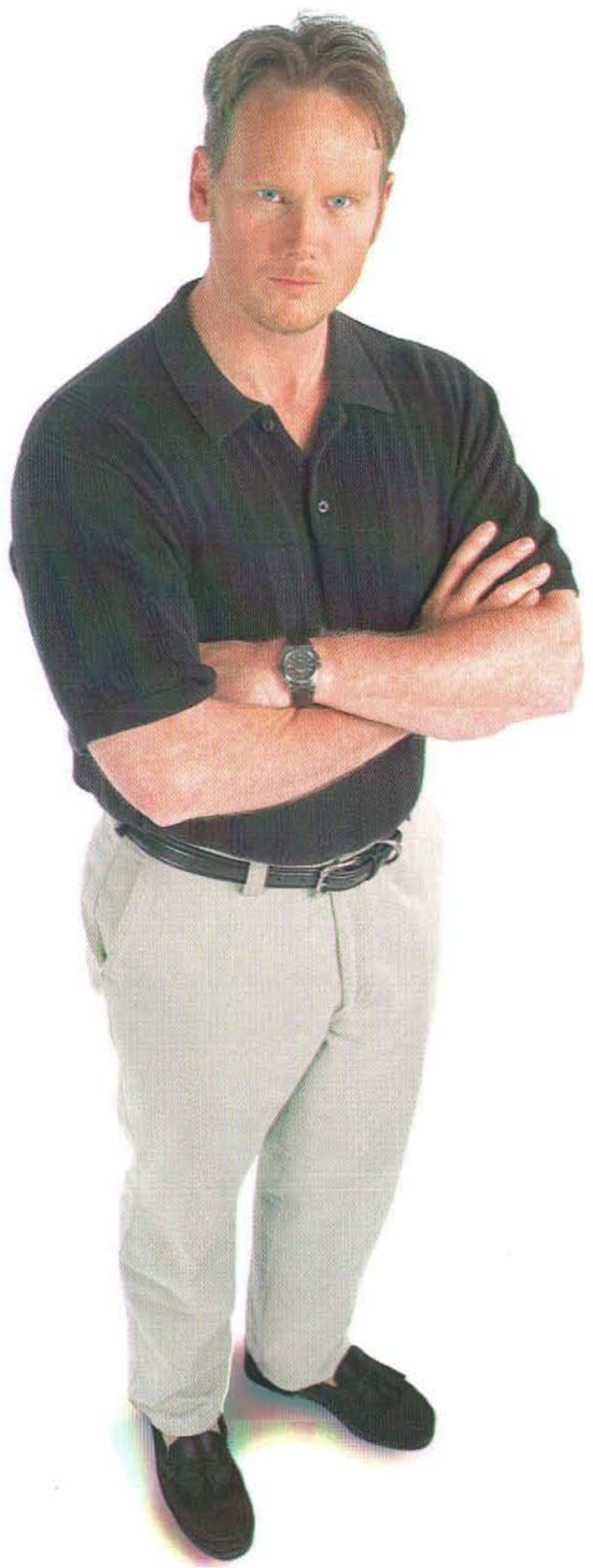
```
myTrackOffset = GetTrackOffset(myTrack);
myMediaTime = TrackTimeToMediaTime(myTrackOffset, myTrack);

// allocate some storage to hold the sample description for the text track
myTextDesc = (TextDescriptionHandle)NewHandle(4);
if (myTextDesc == NULL)
  goto bail;

mySample = NewHandle(0);
if (mySample == NULL)
  goto bail;

myErr = GetMediaSample(myMedia, mySample, 0, NULL,
                myMediaTime, NULL, &mySampleDuration,
                (SampleDescriptionHandle)myTextDesc, NULL,
                1, NULL, &mySampleFlags);
if (myErr != noErr)
  goto bail;

// add actions to the first media sample
switch (theMenuItem) {
  case IDM_MAKE_HYPERTEXT_MOVIE:
    // create an action container for hypertext actions
    myErr =
      QTWired_CreateHyperTextActionContainer(&myActions);
    if (myErr != noErr)
      goto bail;

    // add hypertext actions to sample
    myErr = QTWired_AddActionsToSample(mySample, myActions,
                kHyperTextTextAtomType);
    if (myErr != noErr)
      goto bail;
    break;

  case IDM_MAKE_MEM_DISPLAY_MOVIE:
    // create an action container for wired actions
    myErr =
    QTWired_CreateMemoryDisplayActionContainer(&myActions);
    if (myErr != noErr)
      goto bail;

    // add actions to sample
    myErr = QTWired_AddActionsToSample(mySample, myActions,
                kQTEventType);
    if (myErr != noErr)
      goto bail;
    break;

  default:
    myErr = paramErr;
    goto bail;
}

// replace sample in media
myTrackEditRate = GetTrackEditRate(myTrack, myTrackOffset);
if (GetMoviesError() != noErr)
  goto bail;

GetTrackNextInterestingTime(myTrack, nextTimeMediaSample |
                nextTimeEdgeOK, myTrackOffset, fixed1, NULL,
                &mySelectionDuration);
if (GetMoviesError() != noErr)
  goto bail;

myErr = DeleteTrackSegment(myTrack, myTrackOffset,
                mySelectionDuration);
if (myErr != noErr)
  goto bail;

myErr = BeginMediaEdits(myMedia);
if (myErr != noErr)
  goto bail;

myErr = AddMediaSample(  myMedia,
              mySample,
              0,
              GetHandleSize(mySample),
              mySampleDuration,
              (SampleDescriptionHandle)myTextDesc,
              1,
              mySampleFlags,
              &myNewMediaTime);
if (myErr != noErr)
    goto bail;

  myErr = EndMediaEdits(myMedia);
  if (myErr != noErr)
    goto bail;

  // add the media to the track
  myErr = InsertMediaIntoTrack(myTrack, myTrackOffset,
                myNewMediaTime, mySelectionDuration,
                myTrackEditRate);
  if (myErr != noErr)
    goto bail;

  // update the movie resource
  myErr = UpdateMovieResource(myMovie, myResRefNum, myResID,
                NULL);
  if (myErr != noErr)
    goto bail;

  // close the movie file
  myErr = CloseMovieFile(myResRefNum);

bail:
  if (myActions != NULL)
    (void)QTDisposeAtomContainer(myActions);

  if (mySample != NULL)
    DisposeHandle(mySample);

  if (myTextDesc != NULL)
    DisposeHandle((Handle)myTextDesc);

  if (myMovie != NULL)
    DisposeMovie(myMovie);

  return(myErr);
}
```

Everything in Listing 3 is standard Movie Toolbox stuff that we've seen before, except of course for the application functions QTWired_CreateHyperTextActionContainer (which we'll discuss in the next section) and QTWired_CreateMemoryDisplayActionContainer. This latter function is in fact relatively simple. It builds an atom container that uses the kOperandFreeMemory operand to retrieve the amount of memory that is currently free in the application heap, as well as the kActionTextTrackPasteText wired action to paste the value returned by that operand into the text track. The kActionTextTrackPasteText action, which is new in QuickTime 5, takes three parameters: the text to paste and the beginning and ending locations in the text track to paste the text. Each time we paste a new value into the text track of the memory-display movie, we want to replace *all* of the existing text, so we'll set the second and third parameters to 0 and 0xffff respectively.

There's one undocumented "gotcha" here: in order for kActionTextTrackPasteText to have any effect, we need to explicitly enable text editing on the target text track. We can do this by executing the kActionTextTrackSetEditable wired action (also new to QuickTime 5). This action takes one parameter, which specifies the kind of editing we want to enable. QuickTime currently supports three settings for the editing state of a text track, which the documentation lists like this:

```
#define kKeyEntryDisabled         0
#define kKeyEntryDirect            1
#define kKeyEntryScript            2
```

Come see us at Macworld Expo NY | Booth 1525



My files are important.

Very important.

I only share my files with those I trust.

I trust DAVE®.

Mac to PC, PC to Mac. Cross-platform file and print sharing is too vital to your business to risk. Trust Thursby, the company with 15 years experience. Trust DAVE, the solution with a proven track record. Share files and printers across a network with no barriers. DAVE installs on your Mac with no additional software required for the PC. It's fast, secure and easy to use. Download a free evaluation today!

Trust DAVE®.

Macworld ●●●● $\frac{1}{2}$

www.thursby.com

THURSBY Software

The File Share Folks™

Built for Mac OS X

(Once again, however, these values are not defined in any public header file, so I've added those lines to the file QTWiredActions.h.) The default value kKeyEntryDisabled indicates that no editing is allowed on the text track; key events are passed to the movie controller (which will probably ignore most of them) and any editing actions sent to the track are ignored. The value kKeyEntryDirect indicates that *direct editing* is to be enabled; this means that key events are passed directly to the text track (for instance, key-down events result in the characters being inserted into the text track). The value kKeyEntryScript indicates that *script editing* is to be enabled; this means that editing actions sent to the track are interpreted by the text media handler and executed.

For present purposes, we want to enable script editing on the text track. So we'll execute the code in Listing 4 inside of QTWired_CreateMemoryDisplayActionContainer.

Listing 4: Enabling editing on a text track

QTWired_CreateMemoryDisplayActionContainer

```
myErr = QTNewAtomContainer(theActions);
if (myErr != noErr)
  goto bail;

// add an event atom that enables text editing
myErr = WiredUtils_AddQTEventAndActionAtoms(*theActions,
          kParentAtomIsContainer, kQTEventIdle,
          kActionTextTrackSetEditable, &myActionAtom);
if (myErr != noErr)
  goto bail;

myEditState = EndianS16_NtoB(kKeyEntryScript);
myErr = WiredUtils_AddActionParameterAtom(*theActions,
          myActionAtom, 1, sizeof(myEditState),
          &myEditState, NULL);
```

On every idle event sent to the text track, we set the track to be editable (just before we execute the kActionTextTrackPasteText action). In theory, we could enable editing just once in a frame-loaded event. But the idle event action is simpler to construct and also ensures that we can always paste the current memory value into the text track (since some other action may have disabled editing on that track).

The remainder of QTWired_CreateMemoryDisplayActionContainer is straightforward. It adds an idle event call to the kActionTextTrackPasteText action, using the result of the kOperandFreeMemory operand as the text to be pasted. QuickTime is smart enough to convert the floating-point value returned by kOperandFreeMemory into a string, as expected by kActionTextTrackPasteText. Listing 5 shows the complete definition.

Listing 5: Pasting the amount of free memory into a text track

QTWired_CreateMemoryDisplayActionContainer

```
static OSErr QTWired_CreateMemoryDisplayActionContainer
               (QTAtomContainer *theActions)
{
  QTAtom      myEventAtom = 0;
  QTAtom      myActionAtom = 0;
  QTAtom      myParamAtom = 0;
  short       myEditState;
  UInt32      myPos;
  QTAtom      myParameterAtom = 0;
  QTAtom      myExpressionAtom = 0;
  QTAtom      myOperandAtom = 0;
  OSErr       myErr = noErr;

  myErr = QTNewAtomContainer(theActions);
  if (myErr != noErr)
    goto bail;

  // add an event atom that enables text editing
  myErr = WiredUtils_AddQTEventAndActionAtoms(*theActions,
            kParentAtomIsContainer, kQTEventIdle,
            kActionTextTrackSetEditable, &myActionAtom);
  if (myErr != noErr)
    goto bail;

  myEditState = EndianS16_NtoB(kKeyEntryScript);
  myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, 1, sizeof(myEditState),
            &myEditState, NULL);
  if (myErr != noErr)
    goto bail;

  // add an event atom that displays the amount of application memory currently free
  myErr = WiredUtils_AddQTEventAndActionAtoms(*theActions,
            kParentAtomIsContainer, kQTEventIdle,
            kActionTextTrackPasteText, &myActionAtom);
  if (myErr != noErr)
    goto bail;

  // first parameter: the text to be pasted
  myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, 1, 0, NULL, &myParameterAtom);
  if (myErr != noErr)
    goto bail;

  myErr = WiredUtils_AddExpressionContainerAtomType(
            *theActions, myParameterAtom,
            &myExpressionAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(*theActions, myExpressionAtom,
            kOperandAtomType, 1, 1, 0, NULL, &myOperandAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(*theActions, myOperandAtom,
            kOperandFreeMemory, 1, 1, 0, NULL, NULL);
  if (myErr != noErr)
    goto bail;

  // second parameter: selection range begin: 0
  myPos = EndianU32_NtoB(0);
  myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, 2, sizeof(myPos), &myPos, NULL);
  if (myErr != noErr)
    goto bail;

  // third parameter: selection range end: 0xffff
  myPos = EndianU32_NtoB(0xffff);
  myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, 3, sizeof(myPos), &myPos, NULL);

bail:
  return(myErr);
}
```

On classic Macintosh systems (that is, Mac OS 8 and 9), the movie created by all this code will display the amount of free memory in the application heap (or essentially what you could determine by executing the Memory Manager function FreeMem). On Windows systems and on Mac OS X, where there are no application heaps in the classic-Mac sense, the movie will display less useful numbers. In fact, I'm not exactly sure what the numbers displayed on Windows and Mac OS X represent.

t's 3:27 am, you've hammered out 1237 lines of code...

No one will notice if you take 2 minutes and 25 seconds to jump up and dance.

Artist	Album
Matrix II	All
Men At Work	Don't Censor Me
Men Without Hats	Live Bootleg
Michael Jackson & The Jackson 5	Seltzer - Modern Rock to Settle Your S
Michael W. Smith	Some Kind of Zombie
Michelle Tumes	Underdog
Moby	
Monkees	

Song	Time	Artist	Album
It's Over			
It Is Well With My Soul	3:56	Audio Adre...	Underdog
DC-10	5:12	Audio Adre...	Underdog
Jesus Movement	2:25	Audio Adre...	Underdog
Hands And Feet	3:18	Audio Adre...	Underdog
Let My Love Open The Door	4:08	Audio Adre...	Underdog
Good Life	2:34	Audio Adre...	Underdog
Get Down	3:52	Audio Adre...	Underdog
Underdog	3:15	Audio Adre...	Underdog
	3:30	Audio Adre...	Underdog

GRIFFIN TECHNOLOGY

CYCLONE

KEYSPAN

Sonigistix

Dr. Bott LLC

877.611.2688
503.582.9944
www.drbott.com

Available at:

DEV DEPOT℠
www.devdepot.com

Creating Hypertext Actions

Let's consider now how to construct text movies that contain hypertext actions, like the one shown in Figures 2 and 4. The basic ideas are the same as with constructing wired text movies, except that now we add a text atom extension of type kHyperTextTextAtomType to the end of the text sample. Also, the atom container that's inside of the text atom extension should have the structure shown in **Figure 6**.

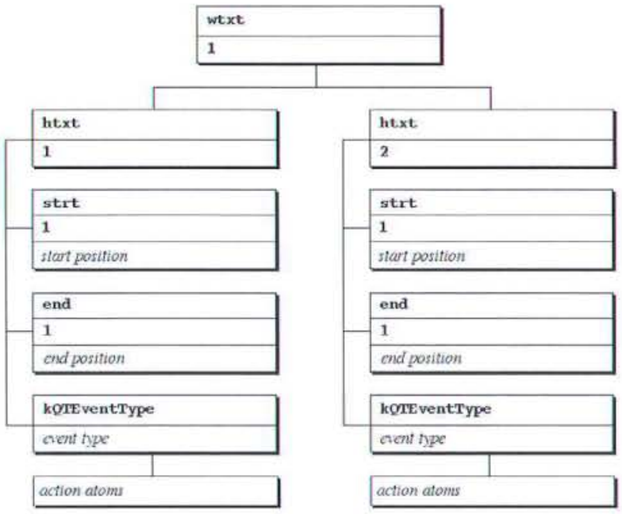


***Figure 6**: The structure of a hypertext text atom*

As you can see, the root atom in a hypertext text atom is of type 'wtxt'. This atom contains one child of type 'htxt' for each hypertext link in the text sample. Let's call this child atom a *hypertext item atom*. A hypertext item atom specifies the information about a single hypertext link in a text sample. We need to specify the starting point and ending point in the sample text for the hypertext link, and we need to specify the event and action atoms associated with that segment of text. So a hypertext item atom needs to contain at least three children, of types 'strt', 'end ', and kQTEventType. In our source code, we'll use these constants:

```
#define kHyperTextTextAtomType          FOUR_CHAR_CODE('htxt')
#define kTextWiredObjectsAtomType     FOUR_CHAR_CODE('wtxt')
#define kHyperTextItemAtomType          FOUR_CHAR_CODE('htxt')
#define kRangeStart                     FOUR_CHAR_CODE('strt')
#define kRangeEnd                       FOUR_CHAR_CODE('end ')
```

There's no need to consider the function QTWired_CreateHyperTextActionContainer in detail. It simply builds the atom container shown in Figure 6, using hard-coded values for the parameter data of the kRangeStart and kRangeEnd atoms. Each of our hypertext item atoms contains three event atoms. The first event atom looks for mouse-clicks on the hypertext link and executes a kActionGoToURL action in response. The second and third event atoms watch for the cursor to enter and exit the hypertext link, changing the color of the text appropriately. Listing 6 shows the code for changing the hypertext color to purple when the mouse enters the first hyperlink. The kActionTextTrackSetHyperTextColor action takes two parameters, which are the index of the hypertext link whose color is to change and the desired new color.

Listing 6: Changing the color of a hypertext link

QTWired_CreateHyperTextActionContainer

```
ModifierTrackGraphicsModeRecord    myGraphicsModeRecord;

myErr = WiredUtils_AddQTEventAndActionAtoms(*theActions,
            myHyperTextAtom, kQTEventMouseEnter,
            kActionTextTrackSetHyperTextColor, &myActionAtom);
if (myErr != noErr)
  goto bail;

myIndex = EndianS16_NtoB(1);
myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, kIndexOne, sizeof(myIndex),
            &myIndex, NULL);
if (myErr != noErr)
  goto bail;

myGraphicsModeRecord.graphicsMode =
            EndianS32_NtoB(ditherCopy);
myGraphicsModeRecord.opColor.red = EndianS16_NtoB(0x9999);
myGraphicsModeRecord.opColor.green = EndianS16_NtoB(0x0000);
myGraphicsModeRecord.opColor.blue = EndianS16_NtoB(0xcccc);
myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, kIndexTwo,
            sizeof(myGraphicsModeRecord),
            &myGraphicsModeRecord, NULL);
```

Take a look at the source file QTWiredActions.c for the complete definition of QTWired_CreateHyperTextActionContainer.

KEY EVENTS

QuickTime 5 introduced a new type of event, the *key event* (of type kQTEventKey), which is issued when keys on the keyboard are pressed. Consider, for instance, the movie shown in **Figure 7**, which displays the ASCII character code of whatever key the user presses. In the case shown here, the user has just pressed the "T" key.

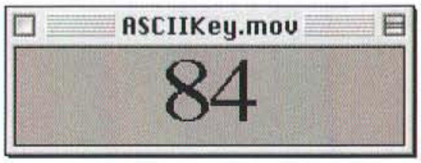


***Figure 7**: A movie that displays ASCII character codes of pressed keys.*

A key event can be used in event atoms just like any of the other event types we've encountered so far. What's interesting about key events is that we can determine not only that the user has pressed a key, but also which particular key was pressed. We do this by inspecting the *event parameters* of the key event, using the new operand kOperandEventParameter. This operand itself takes one parameter, which specifies the index of the event parameter we want to inspect. In the case of key events, we can inspect any one of 5 event parameters:

- Parameter index 1 is the horizontal position of the cursor at the time the key event occurred, in coordinates relative to the text track rectangle.

- Parameter index 2 is the vertical position of the cursor at the time the key event occurred, in coordinates relative to the text track rectangle.

- Parameter index 3 encodes any modifier keys that are down when the key event occurs, using these constants from **Events.h**:

```
enum {
  cmdKey          = 1 << cmdKeyBit,      // 0x0100
  shiftKey        = 1 << shiftKeyBit,    // 0x0200
  alphaLock       = 1 << alphaLockBit, // 0x0400
  optionKey       = 1 << optionKeyBit,   // 0x0800
  controlKey      = 1 << controlKeyBit // 0x1000
};
```

  For instance, if the Control and Shift keys are both down when the key event occurs, then the third parameter would be 0x00001200, or 4608.

- Parameter index 4 is the ASCII character code of the key pressed.

- Parameter index 5 is the *virtual key code* (or *scan code*) of the key pressed. A virtual key code is a value that represents a specific physical key on a specific model of keyboard. As a result, these values are generally less useful than the ASCII character codes.

We can construct the ASCII-display movie in **Figure 7** in exactly the same way we previously constructed the memory-display movie. Instead of issuing the kActionTextTrackPasteText action in response to idle events, we now do so in response to key events. And we get the text to be pasted not from the kOperandFreeMemory operand, but from the kOperandEventParameter operand. Listing 7 shows some of the code we could use to construct the ASCII-display movie. (This code could be substituted for part of Listing 5, for instance.)

Listing 7: Adding a key event

```
QTAtom    myEventParamAtom = 0;
short     myIndex;

// add an event atom that displays the ASCII code of the pressed key
myErr = WiredUtils_AddQTEventAndActionAtoms(*theActions,
            kParentAtomIsContainer, kQTEventKey,
            kActionTextTrackPasteText, &myActionAtom);
if (myErr != noErr)
  goto bail;

// first parameter: the text to be pasted
myErr = WiredUtils_AddActionParameterAtom(*theActions,
            myActionAtom, 1, 0, NULL, &myParameterAtom);
if (myErr != noErr)
  goto bail;

myErr = WiredUtils_AddExpressionContainerAtomType(
            *theActions, myParameterAtom,
            &myExpressionAtom);
if (myErr != noErr)
  goto bail;

myErr = QTInsertChild(*theActions, myExpressionAtom,
            kOperandAtomType, 1, 1, 0, NULL,
            &myOperandAtom);
if (myErr != noErr)
  goto bail;

myErr = QTInsertChild(*theActions, myOperandAtom,
            kOperandEventParameter, 1, 1, 0, NULL,
            &myEventParamAtom);
if (myErr != noErr)
  goto bail;

myIndex = EndianS16_NtoB(4);
myErr = QTInsertChild(*theActions, myEventParamAtom,
            kActionParameter, 1, 1, sizeof(myIndex),
            &myIndex, NULL);
```

Notice that we add a parameter atom (myEventParamAtom) to the operand atom, to specify the desired index for the event parameter we want to retrieve. In this case, we specify the index 4, to obtain the ASCII character code of the key pressed.

It's also possible to use the kOperandEventParameter operand to retrieve event parameters for mouse-related events, such as mouse-enter and mouse-click events. With mouse events, there are only three available parameters, which are the same as the first three parameters of key events: the horizontal and vertical mouse positions, and the modifier keys currently down. In an earlier article, you may recall, we used the kOperandMouseLocalHLoc and kOperandMouseLocalVLoc operands to get these mouse positions. In QuickTime 5 and later, we can instead use kOperandEventParameter, if we so desire.

Bouncing Sprites

In several of the previous articles, we've seen that we can move a sprite around inside a sprite track by altering its matrix, either directly (by setting the sprite's matrix property)

BasePlant

200 classes to enhance PowerPlant™

- AppleEvents
- AppleScript
- Collection Manager
- Data Browser
- Disk Item Filter
- Folder Parser
- Icon Services
- Multiprocessing
- Preferences Manager
- Starting Point
- Window Proxy

And Much More!

$149	*For commercial, in-house development*
$35	*For personal freeware and shareware*

CAFEDERIC.COM

MacAddict's

YOU KNOW WHO YOU ARE

Call 1-888-771-6222
TO GET YOUR FREE ISSUE AND FREE CD-ROM NOW!

TO RECEIVE YOUR FREE ISSUE AND FREE CD-ROM OF MacAddict AT NO RISK!

If you're pleased with what you see, we'll send you 11 more issues and 11 more CD-ROMs (12 in all) for only $29.95. That's a 68% savings off the newsstand rate.

Offer good in U.S. only. Cover price is $7.99. Canada: U.S. $45, includes GST. Foreign: U.S. $58. Prepaid in U.S. funds only. Please allow 6-8 weeks for delivery of first issue.

5UE28

or indirectly (by performing wired actions such as kActionSpriteTranslate). In this section and the next, we'll consider a couple of ways to build on this capability. First, we'll see how to make a sprite bounce around inside of the sprite track's enclosing rectangle; later we'll see how to figure out when two sprites collide with one another. Both of these are ways to give a sprite "physical" properties, so that it appears to react with things around it.

Figure 8 shows (about as well as any static image can, I guess) a sprite bouncing off the track's enclosing rectangle. The sprite is first moving down and to the right; when its bottom edge touches the bottom of the track rectangle, the sprite bounces up and continues to the right. When any other edge of the sprite touches a side of the track rectangle, the sprite does the appropriate thing by moving back away from the side it touched but otherwise continuing in the same direction. (In the event that two of its edges touch two sides of the track rectangle at the same time, the sprite would reverse *both* its horizontal and vertical directions; this would happen when the sprite moves cleanly into a corner of the track rectangle.)

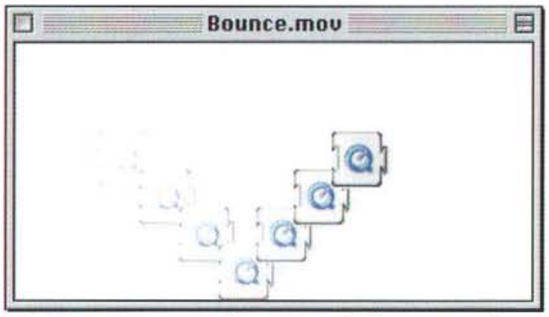


***Figure 8**: A sprite bouncing off the movie edge*

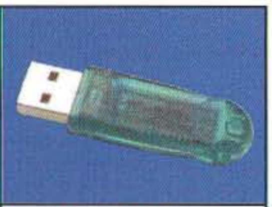
Moving the Sprite

The first thing we need to do is get the sprite moving. To do this, we can simply change the horizontal and vertical positions of the sprite during idle events. Since we're going to be changing the direction of movement when the sprite collides with the track rectangle, we'll maintain two sprite track variables whose values are the number of pixels in the horizontal and vertical direction that the sprite is to be offset during the next idle event. We'll use these variables IDs:

```
#define kXMoveVarID                   2000
#define kYMoveVarID                   2100
```

We can set the initial horizontal and vertical offsets by adding a couple of frame-loaded event actions to the sprite, using our

WIBU-KEY
Software Protection



Best Service
New Technologies
High Quality Products

Can you combine new distribution channels and solid security?

Sure! The solution is WIBU-KEY!

Order Your Test Kit
1-800-986-6578
www.wibu.com

Pay-Per-Use Software Metering
Electronic Software Distribution
Cross Platform Solution for
Mac OS 9, Mac OS X & Windows

Software Protection
License Management
Network Features
Internet Distribution

www.wibu.com

WIBU-KEY is ready for Mac OS X!

GRIFFIN TECHNOLOGIES

USA, Canada: Griffin Technologies, LLC
916 Massachusetts Street, Lawrence, KS 66044
phone: (785) 832-2070 · fax: (785) 832-8787
email: sales@griftech.com · www.griftech.com

Argentina, Australia, Belgium, Canada, Croatia, Denmark, Egypt, Finland, France, Germany, Great Britain, Hungary, Japan, Jordan, Korea, Lebanon, Mexico, Netherlands, Spain, Portugal, Syria, USA

International: WIBU-SYSTEMS AG
Rueppurrer Strasse 52-54 · D-76137 Karlsruhe
phone: +49-721-931 72-0 · fax: +49-721-93172-22
email: info@wibu.com · www.wibu.com

WIBU SYSTEMS
Quality the World Trusts

utility WiredUtils_AddSpriteTrackSetVariableAction (defined in "Wired" in *MacTech*, May 2001):

```
#define kIdleOffset                     2

myErr = WiredUtils_AddSpriteTrackSetVariableAction(
            mySample, kParentAtomIsContainer,
            kQTEventFrameLoaded, kXMoveVarID, kIdleOffset,
            0, NULL, 0);

myErr = WiredUtils_AddSpriteTrackSetVariableAction(
            mySample, kParentAtomIsContainer,
            kQTEventFrameLoaded, kYMoveVarID, kIdleOffset,
            0, NULL, 0);
```

During idle events, we'll move the sprite horizontally by the current value of the kXMoveVarID variable and vertically by the current value of the kYMoveVarID variable. The speed of the moving sprite is determined both by the values of these variables and by the frequency with which we receive idle events (which, you'll recall, is determined by the sprite track's kSpriteTrackPropertyQTIdleEventsFrequency property). For the bouncing sprite movie, we'll tell the sprite media handler to send us an idle event every tick, and (as you can see) we'll offset the sprite in each direction by two pixels during every idle event. We add actions to move the sprite during an idle event like this:

```
myErr = WiredUtils_AddQTEventAtom(mySpriteData,
          kParentAtomIsContainer, kQTEventIdle, &myEventAtom);

myErr = QTWired_AddTranslateByVariablesAction(mySpriteData,
          myEventAtom, kXMoveVarID, kYMoveVarID, NULL);
```

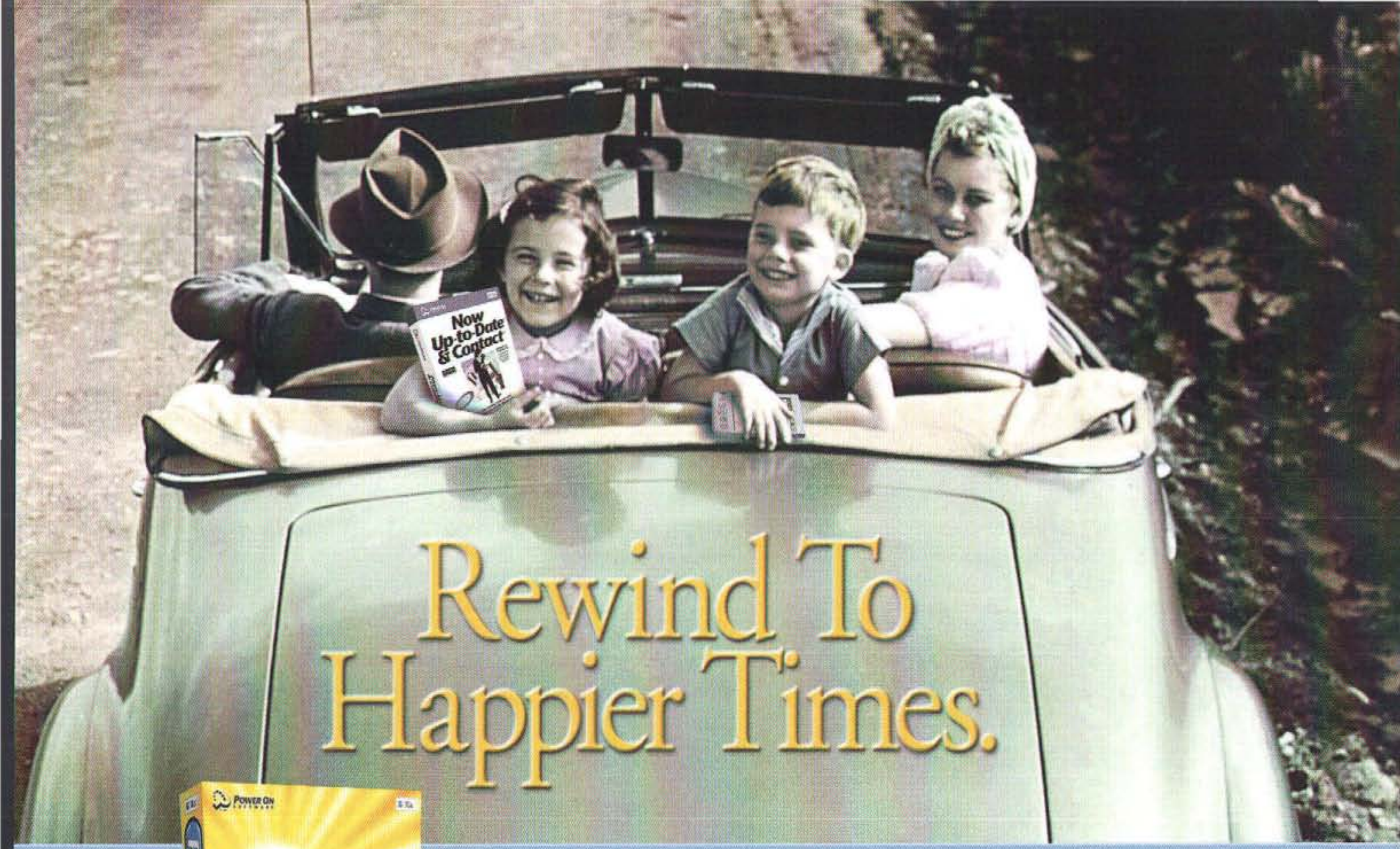
The QTWired_AddTranslateByVariablesAction function adds to the specified event atom (here, myEventAtom) an action atom that translates the sprite relatively, taking the horizontal and vertical offsets from the sprite track variables whose IDs are specified by the third and fourth parameters (here, kXMoveVarID and kYMoveVarID). Listing 8 shows our definition of the QTWired_AddTranslateByVariablesAction function.

Listing 8: Translating a sprite using variable values

QTWired_AddTranslateByVariablesAction

```
static OSErr QTWired_AddTranslateByVariablesAction (
            QTAtomContainer theSprite, QTAtom theParentAtom,
            QTAtomID theXVariableID, QTAtomID theYVariableID,
            QTAtom *theActionAtom)
{
  QTAtom          myActionAtom = 0;
  QTAtom          myExpressionAtom = 0;
  QTAtom          myParamAtom = 0;
  QTAtom          myOperatorAtom = 0;
  QTAtom          myOperandAtom = 0;
  QTAtom          myOperandTypeAtom = 0;
  QTAtomID        myVariableID;
  Boolean         myBoolean;
  OSErr           myErr = paramErr;

  if (theSprite == NULL)
    goto bail;

  // add a translate action atom to the specified parent atom
  myErr = WiredUtils_AddActionAtom(theSprite, theParentAtom,
        kActionSpriteTranslate, &myActionAtom);
  if (myErr != noErr)
    goto bail;

  // first parameter: get value of variable theXVariableID
  myErr = WiredUtils_AddActionParameterAtom(theSprite,
        myActionAtom, kFirstParam, 0, NULL, &myParamAtom);
  if (myErr != noErr)
    goto bail;

  myErr = WiredUtils_AddExpressionContainerAtomType
         (theSprite, myParamAtom, &myExpressionAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myExpressionAtom,
        kOperandAtomType, 0, 1, 0, NULL, &myOperandAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myOperandAtom,
        kOperandSpriteTrackVariable, 1, 1, 0, NULL,
        &myOperandTypeAtom);
  if (myErr != noErr)
    goto bail;

  myVariableID = EndianU32_NtoB(theXVariableID);
  myErr = QTInsertChild(theSprite, myOperandTypeAtom,
        kActionParameter, 1, 1, sizeof(myVariableID),
        &myVariableID, NULL);
  if (myErr != noErr)
    goto bail;

  // second parameter: get value of variable theYVariableID
  myErr = WiredUtils_AddActionParameterAtom(theSprite,
        myActionAtom, kSecondParam, 0, NULL, &myParamAtom);
  if (myErr != noErr)
    goto bail;

  myErr = WiredUtils_AddExpressionContainerAtomType
         (theSprite, myParamAtom, &myExpressionAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myExpressionAtom,
        kOperandAtomType, 0, 1, 0, NULL, &myOperandAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myOperandAtom,
        kOperandSpriteTrackVariable, 1, 1, 0, NULL,
        &myOperandTypeAtom);
  if (myErr != noErr)
    goto bail;

  myVariableID = EndianU32_NtoB(theYVariableID);
  myErr = QTInsertChild(theSprite, myOperandTypeAtom,
        kActionParameter, 1, 1, sizeof(myVariableID),
        &myVariableID, NULL);
  if (myErr != noErr)
    goto bail;

  // third parameter: false (for relative translation)
  myBoolean = false;
  myErr = WiredUtils_AddActionParameterAtom(theSprite,
        myActionAtom, kThirdParam, sizeof(myBoolean),
        &myBoolean, NULL);

bail:
  if (theActionAtom != NULL)
    *theActionAtom = myActionAtom;

  return(myErr);
}
```

Here we add an action of type kActionSpriteTranslate, which requires three parameters: the desired horizontal translation, the desired vertical translation, and a Boolean that indicates whether the translation is absolute or relative. For the first and second parameters, we add expression container atoms that retrieve the value of the variable with the specified ID. (For more information about expression container atoms, see "Wired" in *MacTech*, May 2001.)



In A Bind? Just Rewind!

Remember those good times when everything worked and life was great – before disaster struck? Ever wish you could turn back the clock and undo a software installation gone bad, or retrieve a file that was deleted or overwritten? Have you ever lost a file before it could be backed up and then spent hours recreating it?

By tracking every change you make, *Rewind* can take you back in time by rewinding your Mac to previous systems, preferences, or file versions. The history palette in *Photoshop®* is great, but once you save that file and close it, there's no turning back time to retrieve the three-day old version that your client now prefers — unless you have *Rewind* installed. Better than backup, *Rewind* is quicker, more current, continuous, and complete — and automatic! *Rewind* even offers an emergency start-up mode to get you up and running in seconds, even without a System CD. Best of all, you won't see any performance hit and Rewind needs as little as 5% of your disk space.

You can also ensure accurate memories of your important tasks and deadlines by using *Now Up-to-Date & Contact,* the best choice in contact management and scheduling. If you have to make certain your project is on time, and need instant access to those important contact names and notes, *Now Up-to-Date & Contact* is the product for you. With its support for the *Palm®* and *Visor®*, and the ability to share your calendar with others, no other product offers such ease in keeping track of your contacts and schedule. And if the kids don't remind you of those important dates, our built-in alarms will make sure you never miss one of those those priceless family events.

"...Rewind is a product that will pay for itself in weeks".
MacUser, UK

POWER ON SOFTWARE
The POWER of Productivity.

Available @ DEPOT
<http://www.devdepot.com>
Toll Free: 877-DEPOT-NOW
Outside U.S./Canada: 805/494-9797

MacWEEK.com BEST OF SHOW macworld expo ny 2000

Phone: 800-344-9160 ■ 614-413-4000 Fax-614-413-4100 ■ www.poweronsoftware.com

Detecting Track Rectangle Collisions

So far, so good. But if this were all the wiring we attached to the sprite, we wouldn't get quite the behavior we're looking for. When the movie was first opened, the sprite would begin moving down to the right and would continue moving in that direction forever. We need to add some wiring to figure out when an edge of the sprite hits an edge of the track rectangle and then adjust the direction of movement accordingly. In a nutshell, we want to add some logic that says:

- If the left edge of the sprite is less than the left edge of the track rectangle, then move the sprite back inside the track rectangle and reverse the horizontal direction of travel.
- If the right edge of the sprite is greater than the right edge of the track rectangle, then move the sprite back inside the track rectangle and reverse the horizontal direction of travel.
- If the top edge of the sprite is less than the top edge of the track rectangle, then move the sprite back inside the track rectangle and reverse the vertical direction of travel.
- If the bottom edge of the sprite is greater than the bottom edge of the track rectangle, then move the sprite back inside the track rectangle and reverse the vertical direction of travel.

It's worth pointing out that a more elegant design would use two "if-else" statements here instead of four "if" statements, since the opposite edges of our sprite cannot both be outside the track bounds at the same time. This refinement would, however, require a more complicated wiring. I'll leave that as an exercise for the persnickety reader.

We know the dimensions of the sprite track (since we created it using the constants kIconSpriteTrackHeight and kIconSpriteTrackWidth). So all we really have to learn now is how to figure out the position of the sprite's edges. The sprite media handler supports these four operands, which give us the information we need:

```
enum {
  kOperandSpriteBoundsLeft        = 3072,
  kOperandSpriteBoundsTop           = 3073,
  kOperandSpriteBoundsRight       = 3074,
  kOperandSpriteBoundsBottom      = 3075
};
```

The operand kOperandSpriteBoundsLeft, for instance, returns the left side of the sprite's *bounding box* (the rectangle that encloses the sprite), in the local coordinate system of the sprite track.

In our code, we'll add the side-bounce logic to the sprite by making four calls to a QTWiredActions function QTWired_AddSideBounceToSprite, like this:

```
QTWired_AddSideBounceToSprite(mySpriteData,
      kOperandSpriteBoundsLeft, 1, kOperatorLessThan,
      kXMoveVarID);

QTWired_AddSideBounceToSprite(mySpriteData,
      kOperandSpriteBoundsRight, kIconSpriteTrackWidth,
      kOperatorGreaterThan, kXMoveVarID);
```

```
QTWired_AddSideBounceToSprite(mySpriteData,
      kOperandSpriteBoundsTop, 1, kOperatorLessThan,
      kYMoveVarID);

QTWired_AddSideBounceToSprite(mySpriteData,
      kOperandSpriteBoundsBottom, kIconSpriteTrackHeight,
      kOperatorGreaterThan, kYMoveVarID);
```

The QTWired_AddSideBounceToSprite function is really quite simple, but (as we've grown to expect when building wired actions) a tad lengthy. Since we've built a few wired event handlers already, let's just survey the main points. First of all, we want to install an idle event atom by using our utility function WiredUtils_AddQTEventAndActionAtoms. The action should be a kActionCase action, since we want to ask *whether* a side of the sprite lies outside the track rectangle. As we saw in the previous article, a kActionCase action has a single parameter, whose data is an atom of type kConditionalAtomType. This conditional atom, in turn, has two children, an expression container atom and an action list atom.

The expression container atom needs to test whether the specified side of the sprite lies outside the sprite track rectangle. As you can see, when we call QTWired_AddSideBounceToSprite, we pass in the operand that we need to use to select the side, along with the track limit and the test to perform (for instance, kOperatorLessThan). We build the expression container atom data like this:

```
WiredUtils_AddOperatorAtom(theSprite, myExpressionAtom,
            theTest, &myOperatorAtom);

// first operand: the specified side of the sprite
QTInsertChild(theSprite, myOperatorAtom, kOperandAtomType, 1,
            1, 0, NULL, &myOperandAtom);

QTInsertChild(theSprite, myOperandAtom, theSide, 1, 1, 0,
            NULL, NULL);

// second operand: the specified limit
WiredUtils_AddOperandAtom(theSprite, myOperatorAtom,
            kOperandConstant, 2, NULL, theLimit);
```

The contents of the action list atom are pretty straightforward. Remember that we need to perform two actions: (1) translate the sprite back to the edge of the track rectangle and (2) reverse the direction of travel in the horizontal or vertical dimension. We've already seen how to construct a translate action, so we don't need to repeat that here. We can change the direction of travel simply by negating the value of the variable whose ID is passed to QTWired_AddSideBounceToSprite. We'll do that using another function, QTWired_AddNegateVariableAction, defined in Listing 9.

Listing 9: Negating a sprite track variable

QTWired_AddNegateVariableAction

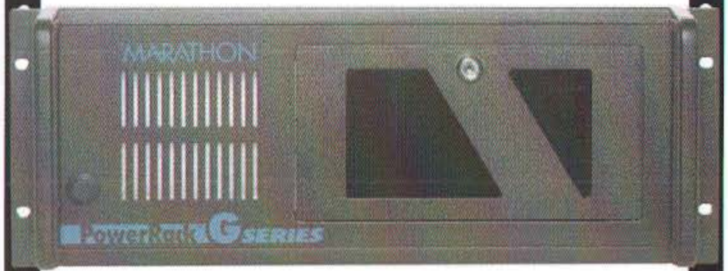
```
static OSErr QTWired_AddNegateVariableAction (
        QTAtomContainer theSprite, QTAtom theParentAtom,
        QTAtomID theVariableID)
{
  QTAtom          myActionAtom = 0;
  QTAtom          myExpressionAtom = 0;
  QTAtom          myParamAtom = 0;
  QTAtom          myOperatorAtom = 0;
  QTAtom          myOperandAtom = 0;
  QTAtom          myOperandTypeAtom = 0;
  QTAtomID        myVariableID;
  OSErr           myErr = paramErr;

  if ((theSprite == NULL) || (theParentAtom == 0))
    goto bail;
  myErr = WiredUtils_AddActionAtom(theSprite, theParentAtom,
              kActionSpriteTrackSetVariable, &myActionAtom);
  if (myErr != noErr)
    goto bail;

  // add parameters to the set variable action: variable ID (QTAtomID) and value (float)
  myVariableID = EndianU32_NtoB(theVariableID);
  myErr = QTInsertChild(theSprite, myActionAtom,
              kActionParameter, 0, (short)kFirstParam,
              sizeof(myVariableID), &myVariableID, NULL);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myActionAtom,
              kActionParameter, 0, (short)kSecondParam, 0,
              NULL, &myParamAtom);
  if (myErr != noErr)
    goto bail;

  myErr = WiredUtils_AddExpressionContainerAtomType(
              theSprite, myParamAtom, &myExpressionAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myExpressionAtom,
              kOperatorAtomType, kOperatorNegate, 1, 0, NULL,
              &myOperatorAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myOperatorAtom,
              kOperandAtomType, 0, 1, 0, NULL,
              &myOperandAtom);
  if (myErr != noErr)
    goto bail;

  myErr = QTInsertChild(theSprite, myOperandAtom,
              kOperandSpriteTrackVariable, 1, 1, 0, NULL,
              &myOperandTypeAtom);
  if (myErr != noErr)
    goto bail;

  myVariableID = EndianU32_NtoB(theVariableID);
  myErr = QTInsertChild(theSprite, myOperandTypeAtom,
              kActionParameter, 1, 1, sizeof(myVariableID),
              &myVariableID, NULL);

bail:
  return(myErr);
}
```

There's nothing too exciting here; this just says: set the value of the variable whose ID is theVariableID to the result of negating the value of the variable whose ID is theVariableID.

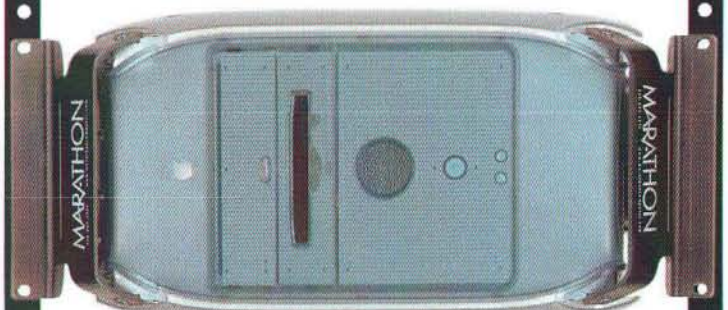
Colliding Sprites

So now we've got sprites bouncing off the walls. Let's get them to bounce off one another as well. In specific, let's create a movie with two sprites, both of which have the bouncing logic that we developed in the previous section. Then, let's add some wiring to make them recoil from one another when part of one sprite touches part of the other. Figure 9 shows two sprites about to collide: the sprite with the new QuickTime logo has just bounced off the bottom and is moving up to the right; the other sprite is moving down to the left. When the sprites collide, we'll reverse the horizontal direction of travel of each sprite unless they are traveling in the same horizontal direction when they collide. And ditto for the vertical directions of travel. This logic gives the sprites a nice feel as they bounce around.

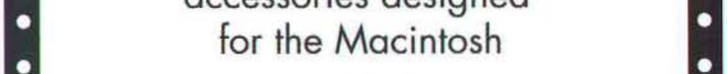
RACKMOUNT YOUR MACINTOSH



PowerRack
Rackmount chassis for PowerMac G3 and G4



G•Rack
Rackmount solution for PowerMac G3 and G4 tower cases

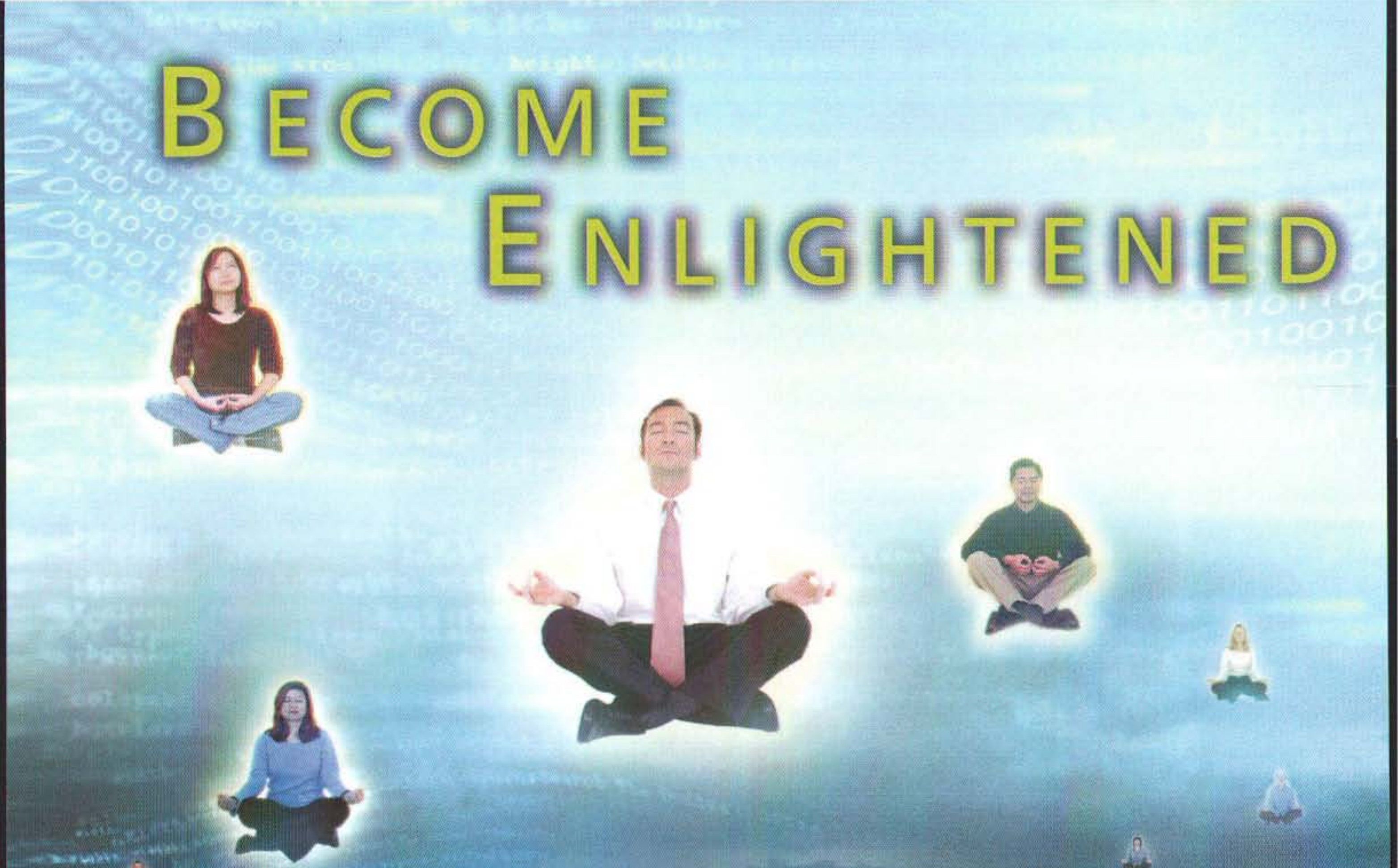


iRack
1RU Rackmount chassis for iMac

Rackmount solutions and accessories designed for the Macintosh

800-832-6326
www.marathoncomputer.com
MacTech@marathoncomputer.com

MARATHON COMPUTER



AWAKEN THE GURU WITHIN YOU

The pressure is on. You've got deadlines to hit, high expectations to meet, and complex technology issues to resolve. And just when the challenges you face on the Web are intensifying, along comes mobile technology.

Now you and your team must create Internet and mobile solutions that raise your organization to a whole new level of efficiency and productivity. Clearly, the path to technological nirvana has never been easy. WEB2001 and Internet+Mobile can show you the way.

Successful Web teams depend on WEB2001 to help them anticipate, evaluate, and implement the latest Web strategy, user experience, and technology. This year, the newly launched Internet+Mobile conference will also prepare you to leverage mobile technology to extend your Internet enterprise.

Gain essential knowledge from the masters of the Web and the pioneers in mobile. Be inspired by fellow gurus from the far reaches of the earth and right next door. And meet the technology providers that will transform your business strategies into reality.

FIVE DAYS. TWO CONFERENCES. ONE EXPOSITION.
ONE ALL-ACCESS PASS. NOW THAT'S GOOD KARMA.

September 4th – 8th, 2001
Moscone Center, San Francisco

JOIN US WWW.WEB2001SHOW.COM

WEB2001
Conference & Exposition

INTERNET+MOBILE
Conference & Exposition

CMP

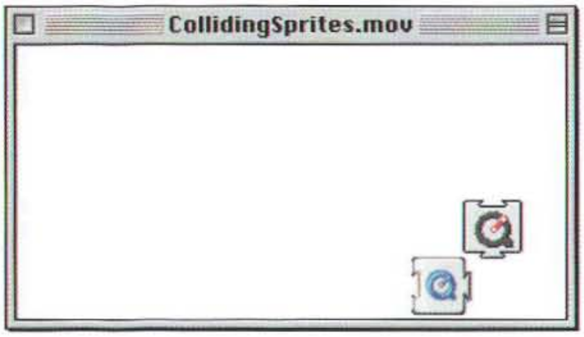


***Figure 9**: Two sprites colliding*

But how do we know when two sprites collide? Well, we know the coordinates of each of the corners of a sprite (using the operands we encountered in the previous section). So a reasonable strategy might be to ask, for each corner of the sprite, whether it lies on top of the other sprite. QuickTime 5 introduced a very useful operand for this, the kOperandSpriteTrackSpriteIDAtPoint operand. This operand takes two parameters, which are the horizontal and vertical coordinates of a point; it returns the ID of the topmost sprite at that point, if any.

In the case where we have just two sprites that can collide, we need to attach the collision logic to only one of the sprites. That sprite (let's call it the collider) can check, on every idle event, whether any of its four corners has come into contact with any part of the other sprite. That is, we'll call kOperandSpriteTrackSpriteIDAtPoint four times, each time passing one of the four corners of the collider. QTWiredActions uses the QTWired_AddCollisionLogicToSprite function (defined in Listing 10) to attach the collision logic to the collider.

Listing 10: Wiring a sprite for collisions

QTWired_AddCollisionLogicToSprite

```
OSErr QTWired_AddCollisionLogicToSprite
                        (QTAtomContainer theSprite)
{
  OSErr          myErr = noErr;

  myErr = QTWired_AddCornerCollisionLogicToSprite(theSprite,
      kOperandSpriteBoundsLeft, kOperandSpriteBoundsTop);
  if (myErr != noErr)
    goto bail;

  myErr = QTWired_AddCornerCollisionLogicToSprite(theSprite,
      kOperandSpriteBoundsRight, kOperandSpriteBoundsTop);
  if (myErr != noErr)
    goto bail;

  myErr = QTWired_AddCornerCollisionLogicToSprite(theSprite,
      kOperandSpriteBoundsLeft, kOperandSpriteBoundsBottom);
  if (myErr != noErr)
    goto bail;

  myErr = QTWired_AddCornerCollisionLogicToSprite(theSprite,
      kOperandSpriteBoundsRight, kOperandSpriteBoundsBottom);

bail:
  return(myErr);
}
```

We won't bother to dissect QTWired_AddCornerCollisionLogicToSprite, as it mostly covers routine ground while setting a new record for length (almost 300 lines of code and comments). The key step is using kOperandSpriteTrackSpriteIDAtPoint, as described above. You should know that kOperandSpriteTrackSpriteIDAtPoint does its work by hit-testing for sprites in the sprite track (probably using SpriteMediaHitTestAllSprites, which we used in an earlier article). So a sprite ID will be returned as the operand's value only if some non-transparent part of a sprite is situated at the specified point.

Our collision logic as developed so far is pretty good, but alas not perfect. Recall that we are testing each of the four corners of the

> INTERNATIONALIZATION & LOCALIZATION

direction International Software Solutions

Intrigued by global opportunities but not sure how to get there? If you are venturing into international markets for the first time, or updating a previous version of your software, we can help you get to market quickly with a fully localized product. Geotopia offers a broad range of internationalization and localization services supporting all aspects of your product, customizable to meet your exact needs. Call us today for a free consultation.

TEL: 800.553.2275 | 801.796.2124 WEB: www.geotopia.com

geotopia™

collider sprite's bounding rectangle, to see if it lies on top of some sprite. If the collider's sprite image at that corner is non-transparent, then the call to kOperandSpriteTrackSpriteIDAtPoint will always return a non-zero value (since the sprite hit test will find that corner, even if the other sprite is nowhere near the collider). We can solve this problem by ensuring that the other sprite has a lower layer property than the collider (so that we get its ID if the other sprite and the collider overlap at the tested point) and by ignoring the collider's ID if kOperandSpriteTrackSpriteIDAtPoint returns it to us.

Another problem arises if the collider's sprite image is transparent at a corner whose coordinates are passed to kOperandSpriteTrackSpriteIDAtPoint. In this case, it's possible for that corner to lie on top of a non-transparent part of the other sprite and hence trigger a hit, even though the sprites do not appear to be touching. **Figure 10** illustrates this possibility. In this case, the collider is the top-right sprite, and its lower-left corner is transparent. kOperandSpriteTrackSpriteIDAtPoint will return the ID of the other sprite, because the collider's lower left corner does in fact lie on top of a non-transparent pixel of the other sprite.

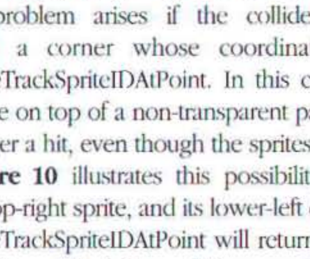


***Figure 10**: Testing a transparent corner of the collider sprite*

I don't see an easy way to solve this problem, but all in all it's a fairly minor one. For most purposes, I suspect, it's good enough to have the sprites recoil from one another using the simple wiring we've developed here. More complex collision algorithms are of course left as exercises for the reader.

CONCLUSION

In this article, we've seen how to add wiring to text tracks and we've investigated a few of the text-related actions and operands added in QuickTime 5. We've also touched briefly on the new key event (of type kQTEventKey) and seen for the first time the technique for retrieving event parameters. Finally, we've added some wiring to allow sprites to collide with the track rectangle and with one another.

With all of this, we've reached the end of our "mini-series" on sprites and wired actions. We'll revisit them in the future, however, especially when we learn how to work with QuickTime VR movies and Flash tracks. In the next article, though, we'll shift gears and move on to consider some new and exciting QuickTime capabilities.

CREDITS

Thanks are due to **Bill Wright** for clarifying some issues with text wiring. MT

Mac OS X
Porting & Development Showcase

Mac OS X
Making the Move

With Mac OS X now HERE Mac developers everywhere have a major need for CARBON and COCOA application porting, AQUA user interface implementation, and DRIVER development services. Toward that end, several high-quality SOFTWARE engineering firms, in association with the APPLE DEVELOPER CONNECTION, are offering these services at very attractive discounts to all ADC Select and Premier members.

The following pages are those vendors that are part of the Mac OS X Porting & Development Showcase.

Add us to your equation

USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II
IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime
Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager
Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI
Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz
Open GL XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3
Players Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP
NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD
Mach Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer
Drivers & Utilities MP3 Players Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP
SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca
Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web
Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager Digital Audio Movies Images USB
FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit
KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime Video
Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager Digital
Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards
BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL
XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players
Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo
Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach
Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer
Drivers & Utilities MP3 Players Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP
SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca
Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web
USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II
IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime
Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager
Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI
Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz
Open GL XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3
Players Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP
NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD
Mach Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer
Drivers & Utilities MP3 Players Sound Manager Digital Audio Movies Images USB FireWire PCMCIA Serial TCP/IP
SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca
Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime Video Digitizer Digital Cameras Web
Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager Digital Audio Movies Images USB
FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards BASIC Stamp Pic II IO Kit
KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL XML QuickTime Video
Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players Sound Manager Digital
Audio Movies Images USB FireWire PCMCIA Serial TCP/IP SMB AFP LDAP NetInfo Modems Bridges PCI Cards
BASIC Stamp Pic II IO Kit KEXT Carbon Classic Coca Objective C Java BSD Mach Kernel Aqua Quartz Open GL
XML QuickTime Video Digitizer Digital Cameras Web Albums Streaming Printer Drivers & Utilities MP3 Players

} +

We consider the factors and provide results.

Sixteen years of experience coupled with the resources and flexibility you need. It's no surprise that our achievements are integrated into many of the products you and your customers have come to rely on every day...**do the math**. Prosoft Engineering, Inc. offers unsurpassed advantages including professional engineers with in-depth knowledge of OS X, partnerships with the top industry leaders, and corporate headquarters located in the heart of Silicon Valley. Prosoft Engineering, Inc., is your source for cost-effective, on-time, custom cross-platform software solutions and services. By uniquely integrating Software Engineering, Quality Assurance, and Project Management, we offer you the solution you need. **Prosoft, it all adds up**.

(925) 426-6100 • www.prosofteng.com • info@prosofteng.com

PROSOFT
engineering inc.

=

Built for Mac OS X

parallel software™

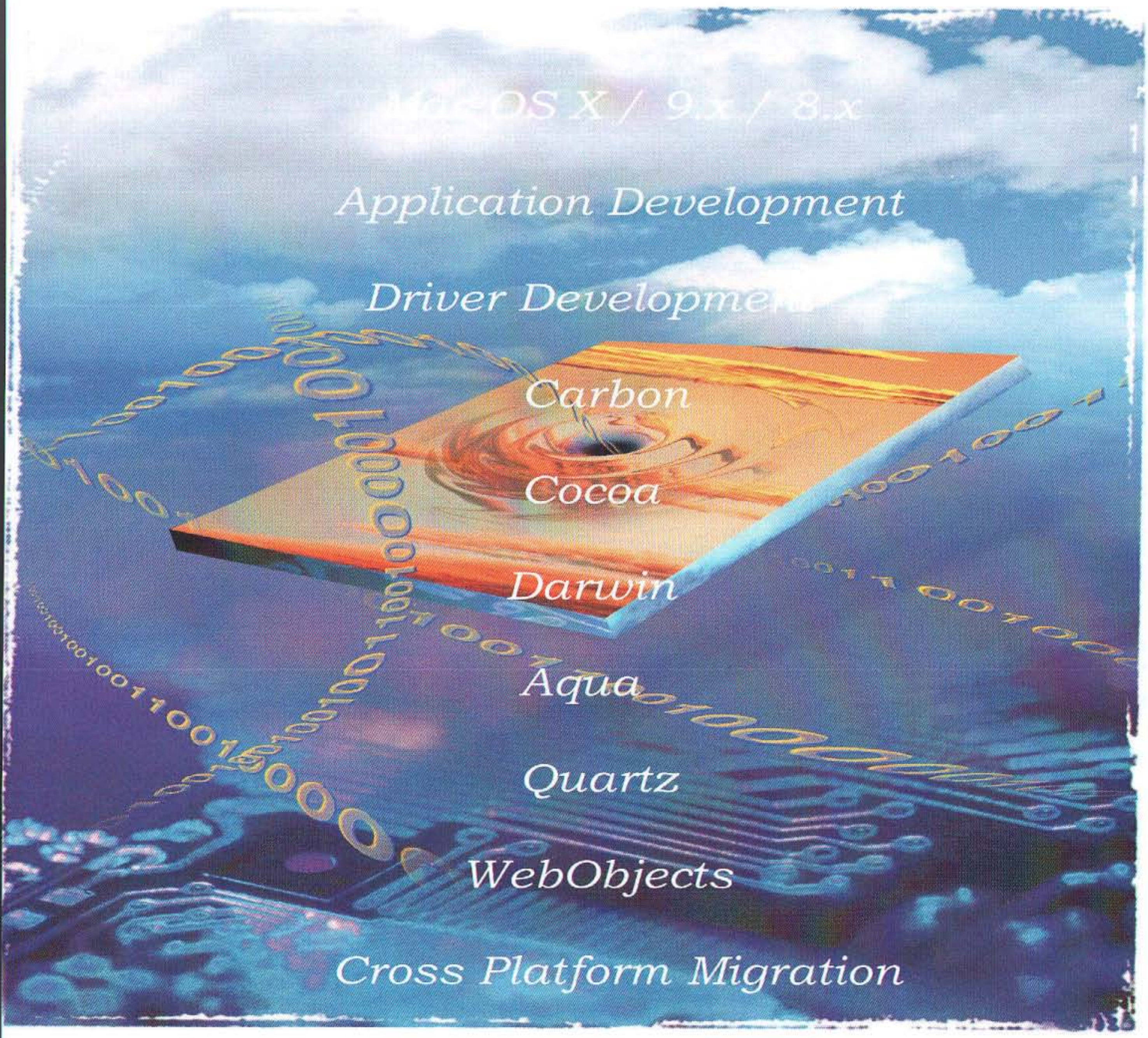
EXPERTS CAN
INTERPRET SIGNS.

OS X MIGRATION EXPERTS

When something unusual occurs, it takes thoughtful advisors to interpret meaning and impact. Moving your software up to Mac OS X? It's not your usual migration—so you need an expert guide who knows how to get you upgraded without getting lost. As the Midwest's largest Apple® developer, Parallel is the OS X transition expert. We'll give uncommon insight to your application planning, development, and successful migration.

Fly by our White Paper, *Mac OS X Migration* at www.parallel.com

Call us at 877-**PARALLEL**



We have the best engineers around.
If it's Mac, we've done it.

- 15 Year Track Record of Delivering Consistent Customer Satisfaction
- Highly Skilled Quality Assurance & Testing Team
- Development Centers Nationwide
- Over 175 Engineers
- Extensive Configuration Testing Facilities

VANTEON™
Delivering Innovative Engineering Solutions

For more information please call 1-800-266-5046, e-mail at mac@vanteon.com or visit us at www.vanteon.com

☐ Eat your vegetables.
☐ Exercise every day.
☑ Port to Mac OS X.
☐ Call your mom.

All of these are good for you.
We can make one of them easy.

Since 1989, The Omni Group has worked with the technologies that have been refined into Mac OS X.

Moving to Mac OS X is a big step for your company, and you need consultants who can help you both plan how best to make the transition and follow whatever path you choose.

That's been our business for years. No matter what kind of product you have, we can get it up under OS X, fast:

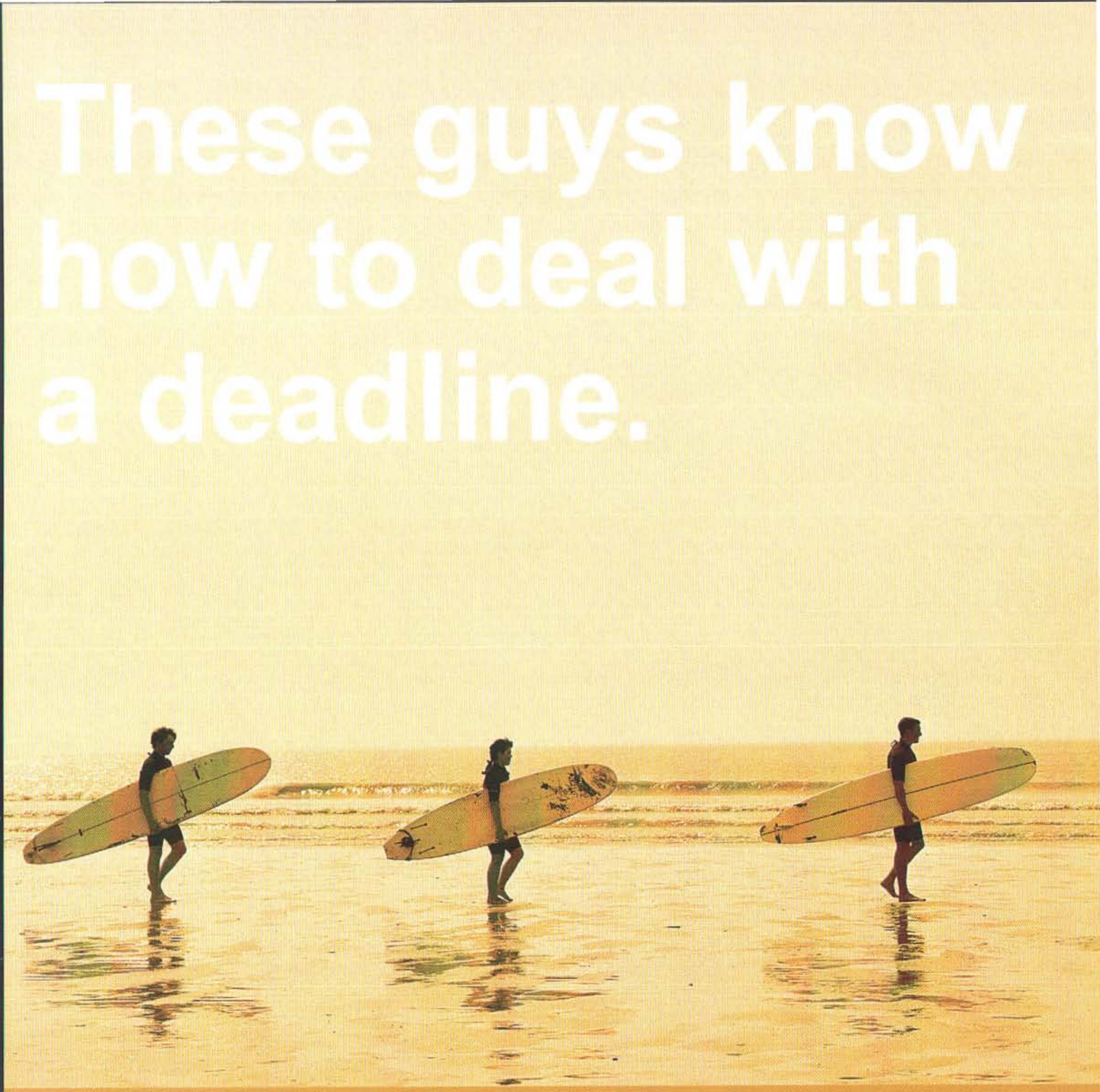
- Real games: We ported id's Doom and Quake games to NEXTSTEP and OS X. Quake 2 took us a week.
- Big apps: We ported Adobe's FrameMaker to NEXTSTEP and Sun's Concurrence to OpenStep/Solaris.
- Big libraries: We ported the Oracle 8 client libraries which Apple ships today in OS X Server.
- Serious drivers: We ported 3dfx's Glide and wrote Voodoo2 and Rendition drivers for OS X Server. We've written new mouse drivers for OS X Server and joystick drivers for OpenStep.
- New apps: We wrote OmniWeb, the only native OS X web browser, and OmniPDF, the native Acrobat viewer for OS X.

Mac OS X is what we do. Let us help you do it, too.

THE OMNI GROUP

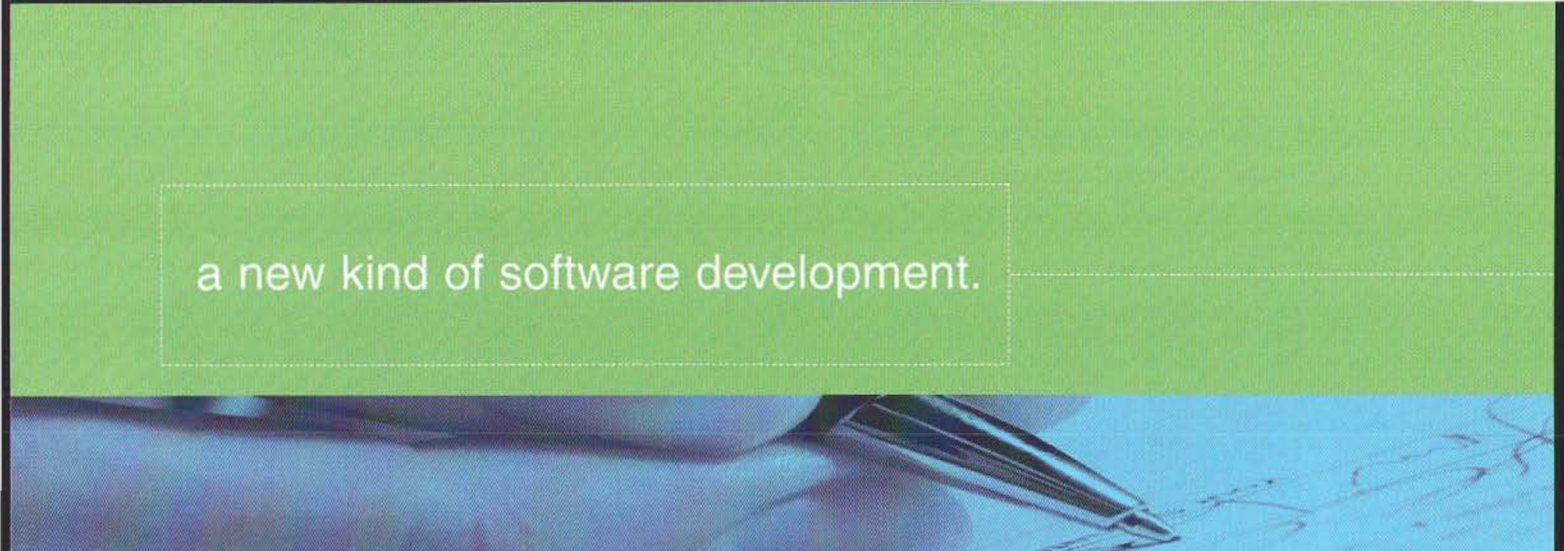
2707 Northeast Blakeley Street
Seattle, Washington 98105-3118
www.omnigroup.com/consulting

sales@omnigroup.com
800.315.OMNI x201
206.523.4152 x201



They hired **Art & Logic.**

software engineering | 877-278-5644 | www.artlogic.com



Without sufficient resources or the right expertise, your project is at risk of shipping late. Atimi guarantees reliable and timely Macintosh software development. You get to market on schedule—and to the satisfaction of both you and your customer. With a wide array of technical expertise, Atimi offers a full range of services:

Classic Mac OS to Mac OS X application porting (Carbon and Cocoa) | Windows to Mac OS application porting | Application development | Driver development (printer, PostScript, USB, Firewire, etc.) | Networking development (wireless and wireline)

Mac

Contact us now and learn more about how Atimi can help you with your project.

Call 604.813.1319 or visit us at www.atimi.com.

Atimi

Software development. On-time.

By Ed Voas, Apple Computer, Inc.

Introduction to Carbon Events

The basic concepts, terms, and philosophy of the Carbon Event Model

Introduction

Without question, the single most important advent in the history of the Mac OS Toolbox is the Carbon Event Model (CEM). "What the heck is that?" you ask. It is your new best friend. It will make it possible for you to write an application in a lot less time, and make your code cleaner and more consistent. It also allows you to tap into the code flow of the Toolbox without patching, and allows Apple to implement new functionality without many undesired side effects. This article gives a high level overview of the terminology and structure of the CEM, and hopefully will demonstrate the power and ease of use it can bring to your applications.

Overview

The CEM is *the* underlying event model for Carbon. This means that classic Toolbox Event Manager routines like WaitNextEvent are built on top of this new model. When calling WaitNextEvent, native Carbon Events are converted into EventRecord structures so your application can deal with them as it always has.

But of course, this new model does much more than serve as a mere foundation for WaitNextEvent — it also exposes a new way of receiving events in your application. Instead of looping around to pick up an event, deciding where it goes and dispatching it yourself, you can instead rely on the Toolbox to do this for you. Also, Carbon Events are much more expressive and expandable than the classic EventRecord structure. This means Apple can add more information to events as the Toolbox evolves.

An important point to note is that this event model isn't just picking up events and doing something with them, but rather it's a whole new inter-object messaging infrastructure. Every component in the Human Interface Toolbox (HIToolbox) ties into Carbon Events somehow. It's quite pervasive.

Another key thing is that the CEM is not something that needs to be adopted all at once. Existing applications can adopt it in stages. We recommend new applications use Carbon Events from the start, as it saves a lot of coding time. I think that once you see what it can do, you'll want to start using it right away (waves hand ala Jedi).

Model Building

There were several compelling reasons to invent a new model.

First, we wanted it to be incredibly simple to write a Carbon application. Most of a basic application involves receiving and delivering events to the appropriate place and handling them with old standbys. For example, what do you do when you receive a click? Typically, you call FindWindow to find out what object was hit by the mouse (window or menu bar). If the click is in the menu bar, you call MenuSelect and act on the result. If the click is in a title bar of a window, you might call DragWindow to move the window around. With CEM, all this default logic is now in the Toolbox, so you don't have to write it anymore. Good. It was a pain anyway.

Second, we wanted to promote an API that encouraged good performance, especially on Mac OS X. Many applications written for past Mac OS releases have used programming idioms and APIs that worked fine on those releases. On Mac OS X, however, they actually work against the system. For example, simple mouse tracking loops will take up close to 100% of the CPU on Mac OS X. With new APIs to replace those tracking loops, the CPU usage in these cases is next to zero.

Third, we have a multitude of messaging models in the Toolbox. We have callbacks, control definitions, window definitions, list definitions, menu definitions, events, and Apple Events. The goal of the CEM is to unify almost all of these into one consistent model. Once you learn it, you know all you need to know in order to listen for new events in the future. Carbon Events subsume everything except Apple Events, which remain the de facto interprocess communication medium.

Fourth, the old EventRecord structure was too limiting. Because of that structure, plus the fact that an event mask was only 16 bits

Ed Voas is the principal architect and developer of Carbon Events. He is also the Manager/Tech Lead of the Carbon High Level Toolbox. When not coding, he is usually practicing Shotokan Karate or attempting to climb a solid 5.5. He can be reached at voas@apple.com. Maybe.

wide, we knew we needed a new transport for events. We needed something that would be able to carry hundreds or thousands of event kinds with plenty of room to grow.

Lastly, this new model supports plug-ins better. Because we can directly dispatch events to windows, controls, and menus, a plug-in can actually get its events directly without necessarily relying on the host application to hand the events to it.

EventRefs: Opaque and Loving It!

The basic building block of the CEM is the EventRef. EventRefs are opaque (virtually all Toolbox entities that end in –Ref are opaque). To get at information contained within an event, you must call accessor functions. For example, to find out when an event occurred, you would call GetEventTime. (We pride ourselves on our original API names.) An event is specified by its *class* and *kind*. For example, a mouse down event has a class of kEventClassMouse, and a kind of kEventMouseDown (Man, we're clever!). There are a ton of new event types defined in the CEM, many more than you've ever seen before on Mac OS.

There are two types of information in an event — information common to all events (such as class, kind, and time), and information specific to a particular event or family of events (such as mouse button, or keyboard modifiers). This type of information is what we term *parameters* of an event. They are accessed via the SetEventParameter and GetEventParameter APIs (how do we think these up?). Parameters are merely generic blobs of data that can be attached to an event. You access them by symbolic names such as kEventParamMouseLocation. The type of data stored in the parameter is represented by a constant such as typeQDPoint. For those familiar with Apple Events, this will seem very similar. Well… OK, identical!

The advantage of this parameterized method is that we can add any number of parameters to any event. We can also do this at any time, meaning that in later versions of Carbon, we can add new parameters to the mouse down event without affecting existing applications. We can even start to store information as different types and automatically coerce them to the original type for existing clients. That's a powerful advantage of opaque types and generic parameter APIs. We dare say they rock.

One other note about events in the New World of Carbon: events are not necessarily one-way. They are also used for two-way communication between objects in the Toolbox. You can ask questions of objects and get responses which get stored as parameters. You do this by sending events directly to objects instead of posting them to the event queue.

Times in an EventRef are expressed differently than in the past as well. In an EventRecord, the when field yielded the time since boot in ticks (via a call to TickCount). EventRef times are also time since boot, but given in seconds as a floating point number. So fifteen and a half seconds after boot is, incredulously, 15.5. The replacement for TickCount when dealing with EventRefs is GetCurrentEventTime. There are macros in the headers that allow you to convert between ticks and the new EventTime type.

The Event Queue

When events come into an application, they are turned into EventRefs and posted onto the application's event queue. There they wait until the application calls an Event Manager API like WaitNextEvent to fetch them.

The event queue is a little different under Carbon than under traditional Mac OS. First, like EventRefs, it is an opaque entity. You cannot directly walk it like you could under traditional Mac OS. Because of this, there is a complete set of routines to allow you to post, search for, and flush events from the event queue.

The event queue is also per-process under Carbon, not global as it is in traditional Mac OS. This means you cannot peek at events destined for other processes, nor can you post events for other processes to receive (except, of course, Apple Events).

There can actually be more than one event queue in your application. The main thread and all cooperative threads share one event queue, which we call the main event queue. Any MPTasks that you might create in your application get their own event queue. This is quite handy if you wish to use Carbon Events as an inter-thread communication medium.

The Event Loop

I mentioned above that events come into an application and get turned into EventRefs. Well, the Event Loop is what takes care of that. When the Event Loop is 'run', it waits for events and converts any that show up into EventRefs and posts them to the event queue. Typically, the Event Loop is run for you inside calls like EventAvail or WaitNextEvent.

The Event Loop is also where your application will block while waiting for an event. It essentially puts your application to sleep until an event (mouse click, Apple Event, etc.) comes into your application. You can think of the Event Loop as a condition variable (for those familiar with such constructs) which you wait on until some event arrives. There's some other stuff that goes on inside here, but we'll cover that later.

It is important to know about the Event Loop and how it works, since it means that there are only certain times the Event Loop is run and events get into your application. If you never run the Event Loop, your event queue will always be empty!

For every Carbon event queue, there is a corresponding Event Loop that is tied to it. With that in mind, like the event queue, the main thread and all cooperative threads share the 'main' Event Loop. MPTasks have their own event loop.

BTW, I'm going to tell you now that the Event Loop term is going to confuse you. It was not the best choice of names, but I couldn't think of anything better at the time.

Event Handlers

For anyone who has used Apple Events, Carbon Event handlers will seem pretty familiar in form and function, with some improvements. These handlers are the means in which your application receives events. You install them by calling InstallEventHandler. You can install handlers onto windows, controls, menus, and the application. The application is a root-level place where events go if not handled by any other handlers.

IOGEAR
KVM • FIREWIRE • USB

USB 2.0 CardBus

GPU202

USB 2.0 PCI Card

GIC250U

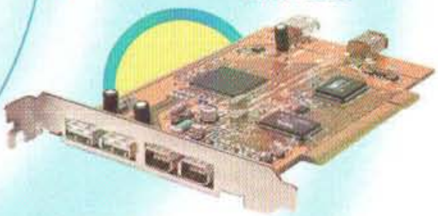
USB 2.0

40 times faster

up to 480 Mbps

USB 2.0 & FireWire PCI Combo Card

GUF220



4 Port USB 2.0 Hub

GUH204

For More Innovation Solutions Please Contact Us:

1-888-999-2836
sales@iogear.com
www.iogear.com

I won't get into where it's best to handle certain events here. That's beyond the scope of this article and best suited for another, more in-depth one. We call that 'job security.'

You install an event handler by calling **InstallEventHandler**:

```
InstallEventHandler(
     EventTargetRef          inTarget,
     EventHandlerUPP         inHandlerProc,
     UInt32                    inNumEvents,
     const EventTypeSpec*    inEventList,
     void *                    inUserData,
     EventHandlerRef*        outHandlerRef );
```

As you can see from the **InstallEventHandler** prototype, you don't actually install handlers onto windows and the like directly. Instead, handlers can only be installed onto *Event Targets*. An event target is quite simply something that can receive events. To install a handler onto a window then, you actually need to get its event target. For example:

```
InstallEventHandler( GetWindowEventTarget(window), … );
```

One other thing you'll notice about **InstallEventHandler** is that it takes an event count and event list in the **inNumEvents** and **inEventList** parameters. This is how you tell the Toolbox what events your handler is interested in. Keep in mind that a) you only will receive events you have registered for and b) you can't receive events that don't make sense to be delivered to a specific target. What that means is that if you have a control in a window, and you install an event handler on it that registers for the **kEventWindowUpdate** event, it will never receive it since those events are always destined for windows.

The **EventTypeSpec** structure used to specify events in the **inEventList** parameter is a simple structure defined thusly:

```
struct EventTypeSpec
{
  UInt32 eventClass;
  UInt32 eventKind;
};
```

The canonical way to define your event list and call **InstallEventHandler** is:

```
const EventTypeSpec kEvents[] =
{
   { kEventClassCommand, kEventCommandUpdateStatus },
   { kEventClassCommand, kEventCommandProcess }
};

InstallEventHandler( GetApplicationEventTarget(),
     MyEventHandlerUPP, GetEventTypeCount( kEvents ),
     kEvents, myUserData, &handler );
```

This registers **MyEventHandlerUPP** to receive two events. The events in this example are command events, which correspond to menu selections. We won't discuss these in this article (remember – job security), but in time you'll see that they're a very powerful way of dealing with menu items and menu selections under the CEM.

Well, that's all fine and dandy, but what does an event handler actually look like? The prototype for an event handler looks like this:

```
OSStatus MyEventHandler(  EventHandlerCallRef inCallRef,
                            EventRef          inEvent,
                            void *              inUserData );
```

We'll discuss the **inCallRef** parameter later, but essentially your handler receives an event and the user data you passed into **InstallEventHandler**. You should return an appropriate **OSStatus** as the function result. This controls how events propagate, as we'll see shortly.

Events get to your handlers by means of the Toolbox dispatcher. This entity is called for you when you are fetching events via **WaitNextEvent** or **RunApplicationEventLoop** (described later). The dispatcher decides where events should go and routes the event to the appropriate entity.

Stack 'Em Up

When handlers are installed, they are actually pushed onto a stack for an Event Target you specify. When an event is sent to that target, the event always goes to the top event handler on the stack. If that handler does not handle the event, it then falls to the next one and so on through all the handlers installed on the target in the reverse order in which they were installed.

This stacked behavior allows your application to override standard Toolbox behaviors. One of the things you can do with the CEM is create a window with a standard Toolbox event handler attached to it. Such a window directly receives events for clicks within its surface, and will automatically handle dragging, clicking the widgets, as well as tracking any controls that are in it. Some of its behaviors are pretty basic, though. For example, when we handle the close box being hit, we dispose the window. In practice, your application might want to do a bit more than that. Installing an event handler pushes a handler on top of the standard handler, allowing your application code to see the event first.

When your handler sees the event, it can choose to handle an event or not. You can also call through to have any handlers below you do their thing first and then post-process the event. You control this by two mechanisms — implicit and explicit propagation.

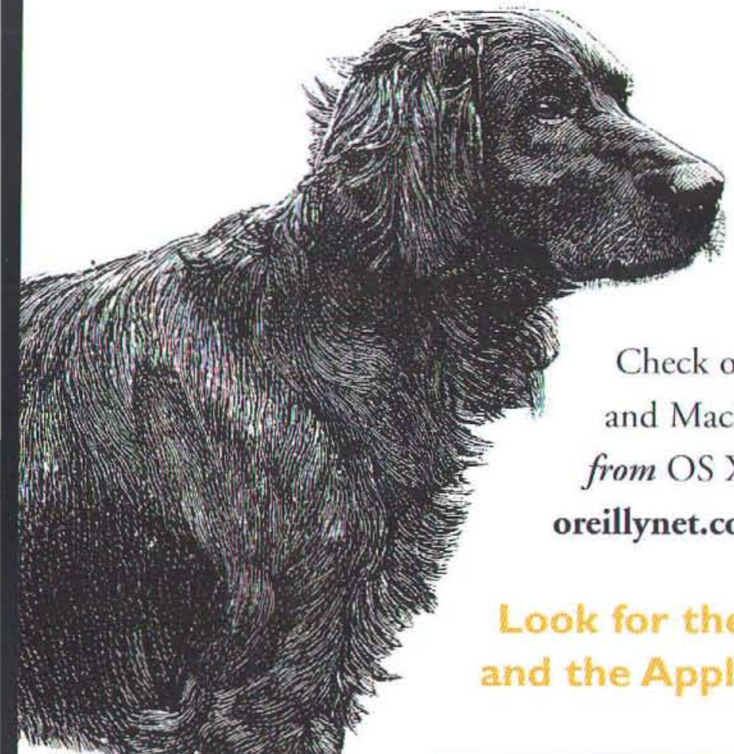
Implicit propagation is the natural order of things. In this situation, event propagation is controlled via the result code of your handler. If your handler returns any other error but **eventNotHandledErr**, it is assumed your handler has either handled the event, or tried to handle it but encountered an error. A completely successful handler would return **noErr**. For any result other than **eventNotHandledErr**, event processing for the current event will stop right after your handler returns.

On the other hand, if your handler returns **eventNotHandledErr**, the event will be propagated onto the next handler in the stack for the current target. It is certainly possible for an event to not be handled at all by any handler. This is perfectly OK, and sometimes actually desirable. Here is an example of a typical event handler:

Listing 1: Typical event handler

```
OSStatus MyEventHandler( EventHandlerCallRef inCallRef,
        EventRef inEvent,
        void * inUserData )
  {
    OSStatus     result = eventNotHandledErr;
    UInt32       theClass, theKind;
    WindowRef window = (WindowRef)inUserData;

    theClass = GetEventClass( inEvent );
    theKind = GetEventKind( inEvent );
```

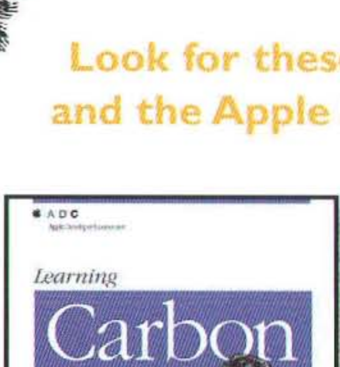
Mac OS X rekindles the spark, O'Reilly fans the flame.

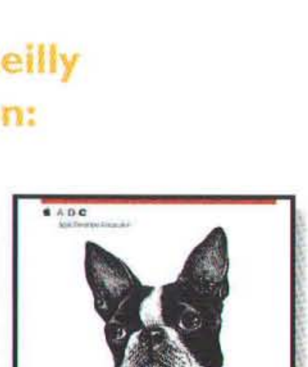


At O'Reilly we solve information problems. From book publishing to online content to conferences—we have what you need to fuel the Mac OS X fire.

Check out our Mac online DevCenter and MacHacks column for the latest information *from* OS X developers *for* OS X developers.
oreillynet.com/mac

Look for these new titles from O'Reilly and the Apple Developer Connection:





800-998-9938
www.oreilly.com
mac.oreilly.com

O'REILLY®

```
    If (   theClass == kEventClassWindow &&
      theKind == kEventWindowDrawContent )
    {
      DrawMyContent( window );
      Result = noErr;
    }

    return result;
  }
```

As you can see from the first line in the function, we always assume we didn't handle the event. That is the best assumption to make. You should only indicate you actually handled an event if you really, honestly, truly handled the event. If you say you handled an event, but you really didn't, you'll feel really guilty later, and you'll be subject to the rules of Karma. In the example above, if we received our draw content event, we set result to **noErr**, indicating we handled the event.

There are times, though, that you want to have the system handlers deal with the event first, and then you can follow up with some neat postlude to the standard behavior. This is where explicit propagation comes into play. In this situation, you want to force handlers below your handler to be called immediately, and then do your follow-up work. You do this via the **CallNextEventHandler** API. When your handler receives an event, you would call through with this API and then do your thing. Typically, you should make sure that **CallNextEventHandler** returns **noErr** (indicating it handled the event) before doing any post-processing. Your handler should also return whatever **CallNextEventHandler** returned for correctness. Here's an example:

Listing 2: Calling through

```
OSStatus MyEventHandler( EventHandlerCallRef inCallRef,
        EventRef inEvent,
        void * inUserData )
  {
    OSStatus  result = eventNotHandledErr;
    UInt32    theClass, theKind;

    theClass = GetEventClass( inEvent );
    theKind = GetEventKind( inEvent );

    If (   theClass == kEventClassControl &&
          theKind == kEventControlDraw )
    {
      result = CallNextEventHandler(
                    inCallRef, inEvent );
      if ( result == noErr )
      {
        // draw 'handles' on the control, which
        // the user can manipulate to resize, drag, etc.

        MyDrawControlHandles( inEvent );
      }
    }

    return result;
  }
```

Now you're saying "Aha! That's what the call ref is for!" Yes, it's strictly used when calling the next event handler, it is basically a blob of data that helps the dispatcher keep track of where it is so it can properly call through. Notice that we only call our function if the event was handled (in this case indicating that the frame was actually drawn). We also return the result of **CallNextEventHandler**, and don't set the result ourselves. The event was handled, so why should we lie about it? Like I said, you'd feel really bad about yourself if you did.

Target Flow

If a target does not handle an event, where does the event go? Well, that depends. In the general case, an event will flow from target to target until handled, based on certain rules imposed by the targets themselves. In general, an event might flow to a control, then it's parent control, then eventually the window and then the application. There are some exceptions, but we'll leave those for some other time.

This target flow can be useful, in that you can handle certain events at any level. For example, a mouse click on a control could be handled at the control level, or possibly the window level. The Toolbox actually uses this to its advantage, using one handler for basic control tracking at the window level, rather than manage many different handlers on every control in a window.

Making It All Go

When writing a Classic Mac OS application, you typically would call **WaitNextEvent** to, well, wait for events. This is the top of the application's event loop (not to be confused with the Event Loop construct in the CEM). WNE also yields time to other applications. If there are no events waiting for your application, it is put to sleep until one arrives or the timeout you pass to WNE expires.

In the CEM, there is a similar routine — **ReceiveNextEvent**. This routine acts almost identically to WNE. The only real difference is that you can control whether you want to actually pull the event from the queue or not. It can also return any event in the event queue while **WaitNextEvent** can only return the events it always has, not any of the new ones. While **ReceiveNextEvent** is a fine, glorious API, you'll probably never call it, except in special circumstances.

Instead, to drive an app written purely to the CEM, your application would call **RunApplicationEventLoop**. This API drives your entire application, and it pulls events off the queue for you and dispatches the events to your handlers automatically. Once called, it does not exit until **QuitApplicationEventLoop** is called. Essentially all of your application code is run via your event handlers.

From Theory To Practice

Now that we've covered most of the basics, let's see what a fully Carbon Event-based application looks like. The simplest case is to use Interface Builder nib files, so that's what we'll base this on.

Listing 3: Basic nib-based Carbon application

```
#include <Carbon/Carbon.h>

int main(int argc, char* argv[])
{
    IBNibRef     nibRef;
    WindowRef      window;

    OSStatus     err;

    // Create a Nib reference passing the name
    // of the nib file (without the .nib extension)
    // CreateNibReference only searches into the
    // application bundle.
    err = CreateNibReference(CFSTR("main"), &nibRef);
    require_noerr( err, CantGetNibRef );

    // Once the nib reference is created, set the
    // menu bar."MainMenu" is the name of the menu bar
    // object.This name is set in InterfaceBuilder
    // when the nib is created.

    err = SetMenuBarFromNib(nibRef, CFSTR("MainMenu"));
    require_noerr( err, CantSetMenuBar );

    //Then create a window."MainWindow" is the name
    // of the window object.This name is set in
    // InterfaceBuilder when the nib is created.

    err = CreateWindowFromNib(nibRef,
              CFSTR("MainWindow"), &window);
    require_noerr( err, CantCreateWindow );

    //We don't need the nib reference anymore.
    DisposeNibReference(nibRef);

    //The window was created hidden so show it.
    ShowWindow( window );

    // Call the event loop
    RunApplicationEventLoop();

CantCreateWindow:
CantSetMenuBar:
CantGetNibRef:
   return err;
}
```

Well, that wasn't so bad, was it? Keep in mind that this application actually does stuff. It creates a menu bar and a simple window using nibs, shows the window and runs the application event loop. There it remains until the app is quit. While running, the app is fully functional — the menus track and the window can be moved or resized, etc. If the user selects quit from the file menu, the app will automatically terminate. We'll see how that works in a short while.

If you have not used Interface Builder before, you should start now. It does an absolutely awesome job of letting you lay out windows and menus. It supports all the latest Toolbox features, and replaces old-style resources with new-style nib files, which are basically XML descriptions of Toolbox UI objects. One of the coolest features it has helps you deal with Aqua HIG specs — when you drag items around, little guide lines appear to help you know what the standard HIG spacing is between items. It's *really* cool.

One other comment about the code above — please notice our use of the standard require macros, which make code much more readable while handling exceptions. This style of error handling was written up a long time ago in Apple's former *Develop* magazine, and we now have these macros in our Universal Interfaces in Debugging.h for everyone to use. I highly recommend them. Consult the header for more details. I mention this because you're going to see it more and more in our sample code and project templates, so it's helpful to be familiar with it.

Periodic Time

RunApplicationEventLoop will block forever while it waits for events. But if it does that, how will you get time to do periodic tasks in your application? Traditionally, applications have relied on null events returned by WaitNextEvent to get such work done. Null events do not exist in the CEM — Event Loop timers are the replacement. If you adopt only one aspect of the CEM, it should be timers. They work with RunApplicationEventLoop-based applications as well as WaitNextEvent-based applications. It's all good.

Timers run at task level — they are not like low-level timer mechanisms which fire asynchronously with respect to your main application flow. You can safely allocate memory and call the toolbox willy-nilly from an Event Loop Timer. They are synchronous in nature, i.e. a timer cannot be interrupted by another timer on the same thread of execution. If you want true asynchronous behavior, you should probably use an MPTask instead, but then you're limited in what you can do (for example, you can't call most of the Toolbox from an MPTask at present).

Timers are run when the Event Loop is run. I mentioned earlier that the Event Loop does some 'other stuff.' This is it. If you don't run your Event Loop, your timers will never fire. This means that timers aren't necessarily a guaranteed heartbeat, but they are more than adequate for most periodic tasks.

Timers are merely callbacks that you install via the InstallEventLoopTimer API:

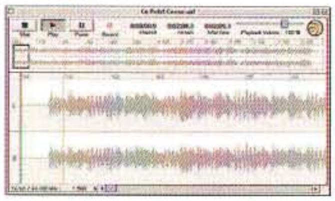
```
OSStatus InstallEventLoopTimer(
        EventLoopRef          inEventLoop,
        EventTimerInterval  inFireDelay,
        EventTimerInterval  inInterval,
        EventLoopTimerUPP     inTimerProc,
        void*                   inTimerData,
        EventLoopTimerRef*  outTimer );
```

Felt Tip
Sound Studio

Record and Edit Audio
with a sound editor designed for the Mac.

Sound Studio will allow you to **make quick edits** with an interface as easy as a **text editor**. Add polish to recordings with fades, normalization, and edits. Create your own mixes. Transform them with effects.

Sound Studio features
- up to 16 bits, 2 channels, and 65 kHz
- up to 2 GB of audio
- AIFF, Sound Designer 2, WAVE, System 7 Sound, and QuickTime import
- edit with sample accuracy
- fade, amplify, normalize, and invert
- delay, echo, reverse, and swap channels
- smooth and emphasize
- resample and pitch shift
- snap to zero crossings, snap to grid



$35
Sound Studio

download free 14-day trial
or order online at
www.felttip.com

QuickTime is a trademark, used under license.
price in US dollars

Felt Tip Software, 807 Keely Place, Philadelphia PA 19128-2326, USA

When installing a timer, you can specify whether you want a periodic timer or a one-shot timer (meaning it fires only once). Passing a non-zero value in the inInterval parameter implies that the timer is periodic, and passing zero means it is one-shot.

Typically, a one-shot timer would be set to fire some number of seconds into the future. (Otherwise, if you want to fire a timer one time immediately, why not just make a function call?) That is what the inFireDelay parameter is for. This parameter can also be used with periodic timers. Basically, inFireDelay is the time interval to wait until we first fire the timer. Once we start to fire, if we have an interval, we will continue to fire at the rate specified in the interval parameter.

The routine we call when the timer fires looks like so:

```
void  MyTimer(EventLoopTimerRef inTimer, void *inUserData);
```

The timer that fired is passed into inTimer, and the data you passed in the inTimerData parameter in InstallEventLoopTimer is passed to your timer in the inUserData parameter.

Inside your timer proc, you can do whatever it is you need to, including removing the timer, or resetting it:

```
OSStatus RemoveEventLoopTimer(
            EventLoopTimerRef     inTimer );

OSStatus SetEventLoopTimerNextFireTime(
            EventLoopTimerRef     inTimer,
            EventTimerInterval  inNextFire );
```

RemoveEventLoopTimer does exactly what you'd expect, it removes the timer and stops it from firing. After calling that API, the timer reference is now invalid and should no longer be used. You need to remove one-shot timers as well as periodic ones — even though they may only fire once, they do not remove themselves automatically. This is mostly because you might want to reset the timer after it fires.

SetEventLoopTimerNextFireTime lets you reset a timer. This is typically used for a one-shot timer. For example, let's say if the user doesn't move the mouse in the next 10 seconds, you're going to quit your application (that's just the way you are). You would install a one-shot timer with an inFireDelay of 10 seconds, and an interval of 0. Each time the user moves the mouse, you would reset the timer with SetEventLoopTimerNextFireTime, passing 10 into inNextFireTime. If the timer fires, you set your quit flag and remove the timer.

You can also use SetEventLoopTimerNextFireTime with periodic timers if you wish, possibly postponing some sort of periodic action for a number of seconds. This API does exactly what it says, it sets the next time the timer will fire. This applies to one-shot as well as periodic timers. If you have a timer which fires every second, and you set the next fire time to 10 seconds from now, the timer will be dormant for 10 seconds, and then start firing every second after that again.

Performance On Mac OS X

All this talk about timers reminds me of performance issues, particularly on Mac OS X. The goal which every application should strive for on Mac OS X is to use 0% of the CPU when idle. That's right, zero. If your app is not foreground, there should be no activity. This has a substantial impact on overall system responsiveness and performance. This means that 100% of the CPU is available for meaningful tasks, such as downloading files, or rendering images. It also means that the VM working set is substantially less, since we don't have to keep pages hot for applications in the background. If you are polling in the background, you are stealing resources from the foreground application, making things slower for quite possibly no good reason.

Some people respond to this by saying "Wait a second you crackpot, I thought Mac OS X was preemptively scheduled. It shouldn't matter if I spin the processor." While this is true, remember that we are all sharing the CPU on X, and if two apps are spinning as fast as they can, they will only get about 50% of the CPU. Three app and you get 33%. 10 apps and you can now only hope for 10% of the CPU. If nine of those apps are just checking to see if a file has appeared in some folder, and one is trying to do real work and download a file, your download will be 10 times slower than it could be. That's pretty significant. Yes, even in this brave new world, we need to be good citizens. We can't stop another process from running like we can in traditional Mac OS, but we can sure slow it down.

Now, don't get me wrong — there certainly are classes of application that must get idle time constantly: browsers displaying animated GIFs, clock applications, music players which display frequency meters, etc. The goal though is to not take up any more idle time than is necessary. If you are displaying an animated GIF that displays each frame every second, then you only need a one second idle heartbeat in your application. If you are displaying no animated GIFs, you might not need any time. The trick is to only get the time you absolutely need.

Reaching this goal of zero CPU usage may seem impossible, but the CEM goes a long way to help you reach that goal. We offer several means of accomplishing that — Timers, better tracking loops, and new events to notify you when certain things change. You'll find that getting down to zero CPU usage is not impossible at all.

User Focus

One concept we are introducing with the CEM is that of *user focus*. Put simply, the user focus is the Event Target where keyboard events are sent. You can think of it as the extension of the current keyboard focus concept in the Control Manager. Menu commands are also sent to the user focus, but we'll look at those in the next section.

The user focus is the combination of the current *user focus window* and any currently focused control in that window. The user focus window is normally defined as the currently active document window. If no controls are focused in that window, the window is the current User Focus target, and keystrokes are sent there. If there is a focused control in the window, the control is the current User Focus target, and it receives keystrokes.

When the user clicks on a document window, the Window Manager changes the user focus window to the newly selected window automatically. If the user clicks in a floating window, the user focus window does not normally change. However, if the user clicks in an editable text field in the floating window (or any control that wants focus on mouse clicks), the Toolbox will switch the user focus to the floating window for you. Yes, the Toolbox makes the rash assumption that you'd like to let the user actually type into that swell text field you put in your floating window. If the user clicks in the body of another floater (but not in a focus-on-click control), the user focus is reset to the currently active document window. Likewise, if the user clicks in the currently active window (or some other document window) while a floater has focus, the user focus is reset to the window that was clicked.

It is possible to alter where keystrokes go by either changing the keyboard focus via the existing Control Manager APIs such as AdvanceKeyboardFocus, or by using the new SetUserFocusWindow API. This API can divert focus away from the default of the active window. This is how the floating window behavior above is implemented. You can also reset the focus to the Toolbox's standard choice with SetUserFocusWindow by passing (WindowRef)-1L for the window ref you pass in.

Commands

Menu commands have been supported since Mac OS 8.0. Essentially, they represent a position-independent way of identifying menu items or menu selections. No matter where your menu item happens to be, the command ID is always constant. Carbon Events extends the concept of menu commands to include controls as well. Both menus and controls use Carbon events to indicate that the user has manipulated the object. A menu selection or a click on a control will generate a Carbon event of class kEventClassCommand and kind kEventCommandProcess containing the command ID associated with the command.

The Carbon event manager packages the command ID generated by a menu or control into a new object called an HICommand. This structure contains information about the command, such as where it was generated from (menu and item number), as well as the actual command ID. Currently, control information is not contained in this structure, but that will likely change in the future. The Toolbox uses this to determine how to propagate commands through the containment hierarchy. An HICommand that was generated from a menu will get sent first to the menu it came from, and then to the user focus, and so on down the chain. An HICommand generated from a control merely starts at the control it came from and down the containment hierarchy as any other control event would.

Commands are essential to a Carbon Event-based application. It is, in fact, a menu command that allows a standard Carbon Event-based app to quit. As long as the Quit menu item has the standard menu command of kHICommandQuit, your application will terminate normally. This is how the nib-based example we looked at earlier quits. When the kHICommandQuit event is sent from the menu selection, it eventually reaches the standard application-level Toolbox handler, where the Toolbox responds by sending a kEventQuitApplication event to the application target. If the kEventQuitApplication event is not handled, the Toolbox receives it in the same application-level Toolbox handler and call QuitApplicationEventLoop. This terminates the call to RunApplicationEventLoop, and eventually exits the program. If you happen to be a WaitNextEvent application, you won't quit automatically, but you could intercept the command event with an application-level event handler and trigger your normal quit process.

Commands are also important for menu item enabling. Whenever a menu is about to be displayed, the Menu Manager attempts to ensure each item is in the correct state, i.e. it is enabled or disabled, has the right text, and so forth. It does this by sending kEventCommandUpdateStatus events to the current user focus for each item in the menu. This is the time for the user focus to examine its current state and adjust its menu items as appropriate. For example, if the Edit menu was about to be displayed, you might want to enable the Cut and Copy menu items in your Edit menu if you have an active selection. If your application needs to change the name of the item (for example, to make the Undo item say "Undo Typing"), you can do that at the same time. Essentially, you should make sure the menu item which corresponds to the command sent as part of the kEventCommandUpdateStatus event is set up correctly for display and selection by the user.

Let's look at a small example of a handler that deals with kEventCommandUpdateStatus events.

Listing 4: A command status handler

```
OSStatus
MyCommandStatusHandler( EventHandlerCallRef inCallRef,
                        EventRef            inEvent,
                        void *              inUserData )
{
  OSStatus      result = eventNotHandledErr;
  HICommand   cmd;
  MyObject*   object = (MyObject*)inUserData;

  //The direct object for a command event is the HICommand.
  // Extract it here and switch off the command ID.

  GetEventParameter( inEvent, kEventParamDirectObject,
      typeHICommand, NULL, sizeof( cmd ), NULL, &cmd );

  // Only deal with menu commands.The Toolbox will eventually send update status
  // events for control commands as well, so we should limit our scope to menus if
  // that's all we care about.

  if ( (cmd.attributes & kHICommandFromMenu) != 0 )
  {
    //You'll note below that we enable and disable
    // by menu ref and index rather than the command
    // ID.This is because it is more efficient to do
    // so, since access by command ID requires the
    //Toolbox to search all menus. Either way is fine,
    // but I thought I'd point that out.

    switch ( cmd.commandID )
    {
      case kHICommandCopy:
      case kHICommandCut:
        if ( object->HasSelection() )
        {
          EnableMenuItem(  cmd.menu.menuRef,
                           cmd.menu.menuItemIndex );
        }
        else
        {
          DisableMenuItem( cmd.menu.menuRef,
                           cmd.menu.menuItemIndex );
        }
        result = noErr;
        break;
      case kHICommandUndo:
        if ( object->CanUndo )
        {
          EnableMenuItem(  cmd.menu.menuRef,
                           cmd.menu.menuItemIndex );
          SetMenuItemText( cmd.menu.menuRef,
                           cmd.menu.menuItemIndex,
                           object->GetUndoString );
        }
        else
        {
          DisableMenuItem( cmd.menu.menuRef,
                           cmd.menu.menuItemIndex );
          SetMenuItemText( cmd.menu.menuRef,
                           cmd.menu.menuItemIndex,
                           CFSTR( "Can't Undo" ) );
        }
        result = noErr;
        break;
    }
  }
  return result;
}
```

As you can see from the listing, we look for the cut, copy, and undo commands in our handler. For cut and copy, we merely enable or disable the menu item depending on whether our object has a selection. For the undo command, not only do we enable or disable the item, but we also change the text of the menu item as appropriate. In each case, we return noErr to let the event system know this handler handled the event.

This very mechanism allows the editable text controls in the Toolbox to update the Edit menu appropriately depending on the current selection. For this reason, it is important that the Edit menu in your application uses the standard cut, copy, paste, etc. commands that are currently defined in CarbonEvents.h. If the Edit menu does not work right for a standard Toolbox text field, this is probably what's wrong. We used to have a really slimy mechanism in place to do this behind the app's back. On Mac OS X, that mechanism is gone, so we need a little cooperation from your application so we can all play in the same menu sandbox.

One thing to keep in mind — do not assume a command came from a menu! Always check the attributes of the command to make sure the kHICommandFromMenu bit is set. If not, don't try to access the menu.menuRef or menu.menuItemIndex fields. Remember that commands can come from controls, and Apple will be adjusting this structure in the future to accommodate that.

MODALITY

There are times in your application where you need to put up a modal dialog, i.e. it disallows clicks in other windows. Usually, when in this state, the menu bar is disabled except for a few choice menu items. In the past, you would have had to either use ModalDialog, or roll your own modal event loop.

The CEM offers a new way to go about this: RunAppModalLoopForWindow. Like RunApplicationEventLoop, once called, your application stays inside this API until a corresponding call to QuitAppModalLoopForWindow is made. Once you enter this state, the menu bar automatically disables. Depending on what command handlers you have installed for the window, you can enable certain menu items within your modal context. For example, even though the menu bar is disabled, if you tab into a text field, the Edit menu will react as it should, handling kEventCommandUpdateStatus events that are sent to it.

Usually when you implement something like this, you install an event handler for the window to capture command events. Inside this handler is where you would end up calling QuitAppModalLoopForWindow. Listing 5 shows an example of creating a dialog from a nib file and running it modally. We are essentially asking the user a yes or no question, and the result of the interaction with the user determines whether we return true or false.

Listing 5: Running a dialog modally

```
Boolean
ConfirmActivateSelfDestruct()
{
  IBNibRef            nibRef;
  EventTypeSpec     cmdEvent =
          { kEventClassCommand, kEventCommandProcess };
  WindowRef         window;
  OSStatus            err;
  EventHandlerUPP   handler;
  ConfirmData       data;
  Boolean           result = false;

  // Get the nib reference, create the window from it, and release the nib reference.

  err = CreateNibReference( CFSTR( "Main" ), &nibRef );
  require_noerr( err, CantOpenNib );

  err = CreateWindowFromNib( nibRef,
           CFSTR( "ActivateSelfDestruct" ), &window );
  require_noerr( err, CantCreateWindow );

  DisposeNibReference( nibRef );


  // Install an event handler to listen to command events for the window.
  // This is the only type of event we care about!

  // We pass data needed by our handler in to our call to InstallWindowEventHandler.
  // Here we pass the window ref, as well as the magic boolean which tells us
  // whether the self-destruct mechanism should be activated.

  data.confirmed = false;
  data.window     = window;

  handler = NewEventHandlerUPP( SelfDestructHandler );
  InstallWindowEventHandler( window, handler, 1,
        &cmdEvent, &data, NULL );

  // Move the window into position and show it

  RepositionWindow( window, NULL,
        kWindowAlertPositionOnMainScreen );
  ShowWindow( window );

  // Now run the window modally. We will remain inside
  // RunAppModalLoopForWindow until the Quit... call is made
  // in our EraseDiskHandler routine.

  RunAppModalLoopForWindow( window );

  // We're done! Get our result, dispose the window, and exit

  result = data.confirmed;

  DisposeWindow( window );
  DisposeEventHandlerUPP( handler );

  return result;

CantCreateWindow:
  DisposeNibReference( nibRef );

CantOpenNib:
  return result;
}
```

If you look closely, you will see it's extremely similar to our main function in our basic application we looked at earlier. The only real difference is we have installed a window event handler on the window we create to handle command events. The next listing shows a simple example of such a handler:

Listing 6: Our self-destruct modal dialog handler

```
OSStatus
SelfDestructHandler(EventHandlerCallRef inCallRef,
                    EventRef             inEvent,
                    void*                  inUserData)
{
  HICommand     cmd;
  OSStatus        result = eventNotHandledErr;
  ConfirmData*  data = (ConfirmData *)inUserData;

  //The direct object for a 'process command' event is the HICommand.
  // Extract it here and switch off the command ID.

  GetEventParameter( inEvent, kEventParamDirectObject,
       typeHICommand, NULL, sizeof( cmd ), NULL, &cmd );

  switch ( cmd.commandID )
  {
    case kHICommandOK:

       //The OK button was clicked. Store true into our magic boolean.

       data->confirmed = true;

       // Stick a fork in us, we're done.Terminate the call to
       // RunAppModalLoopForWindow and return noErr to indicate
       // we handled this command.

       QuitAppModalLoopForWindow( data->window );
       result = noErr;
       break;

    case kHICommandCancel:

       //The user clicked cancel. Our data was passed in to us with
       // false set, but since we don't want to risk the ship based on
       // my assumptions, we set it to false explicitly! We then quit
       // the modal loop and return noErr to indicate handled this event.

       data->confirmed = false;

       QuitAppModalLoopForWindow( data->window );
       result = noErr;
       break;
  }

  return result;
}
```

If you're wondering where these commands are coming from, they are being read in as part of the nib file. For any control in a nib, you can set a command ID for it. When the nib is read in and the controls are created, any command ID you specified will automatically be set for the control. In our example, if the kHICommandCancel or kHICommandOK commands are received, we call QuitAppModalLoopForWindow, which terminates our modal loop.

THE DIALOG WHAT?

The interesting thing with the examples in the last section is that you'll notice that with the combination of nibs and commands, the Dialog Manager is obsolete. The only thing we didn't cover is being able to access controls by control ID. This is something new in Carbon that allows you to get at your controls with a unique identifier. For dialog items using the Dialog Manager you would have typically used GetDialogItem or GetDialogItemAsControl, passing the dialog item number of the item you were interested in. That has a major failing in that if you reorder your items, you need to change all your constants, as the dialog item number is just an index into the dialog item list. Control IDs allow position independence — it doesn't matter where the control is in the control hierarchy for a window, you can easily find it with the control ID. If you move it, you'll still find it with the same code. Interface Builder lets you assign IDs to controls so that when you instantiate an object from a nib, the control IDs are all set to go.

The other thing the Dialog Manager has historically kept track of is the default and cancel items of a dialog. The default item is the item that would be 'hit' by pressing enter or return, and the cancel item is the item which would be 'hit' by pressing command-period or escape on the keyboard. To replace that, we have introduced SetWindowDefaultButton and SetWindowCancelButton. The standard Toolbox event handlers for windows check for default and cancel button 'hits' like the Dialog Manager would. Again, Interface Builder lets you specify that a button is the default or cancel button.

All of this means you can now lay out dialogs with Interface Builder (which is really easy!), and run them with the Carbon Event Model, and you don't have to worry about the Dialog Manager's idiosyncrasies from 1984.

OW! MY BRAIN HURTS!

Wow. That was a lot of information! But I think we covered all of the basic aspects of the Carbon Event Model and how they work. Now you have a good foundation of knowledge upon which you can start building. In future articles, we can start moving into concrete examples and more esoteric aspects of all this.

As you can see, it's a very exciting addition to the Toolbox. We see it as the piece that has been missing for too long which will allow the Toolbox to grow into something really amazing as we move forward. Any new functionality we add will generally involve some aspect of this new event system, so it's important to understand it.

As was evident from the examples, you can write a lot of application in very little code. In the future, we will be able to reduce that code even further, possibly to as little as 3 lines of code to start up an application! We will also be able to add more functionality for free while still allowing your applications to alter behaviors. I encourage you all to start playing with this stuff and see what it can do.

It's the beginning of a whole new era for Carbon application programming!

*Special thanks to **Eric Schlegel**, **Guy Fullerton**, and **Matt Ackeret** for reviewing this article.* MT

List of Advertisers

4D	1
AD Software	19
AEC	25
Aladdin Knowledge Systems	55
Aladdin Systems	9
Appgen	11
ASC Tech	14
Automated Solutions	5
Bowers Development	34
Cafederic	67
CMP Media	74-75
DevDepot	28-29
Dr. Bott	51, 65
edition.net	39
eSellerate	47
FairCom Corporation	7
Farallon	77
Felt Tip	91
Fetch Softworks	8
Full Spectrum	38
Geotopia	76
IDG World Expos	56-57
IOGear	87
Jiiva, Inc.	15
LexTek	39
Mac OS Radio	35
MacAddict	68
Marathon	73
Mathemaesthetics	45
Metrowerks	BC
MindVision	33
Netopia	21
O'Reilly	89
Onyx Technologies	48
OpenBase International	13
Paradigma	31
Pitsco-Lego Dacta	37
Power On Software	71
PrimeBase	53
Quest	17
REAL Software	49
Scientific Placement	19
Stone Tablet	27
SuSE	IBC
Tera Global Communications	IFC
Thursby	63
UNI SOFTWARE PLUS	61
WIBU Systems	69

List of Products

4D and WebSTAR • 4D	1
Appgen Moneydance • Appgen	11
AppMaker • Bowers Development	34
BasePlant • Cafederic	67
CodeWarrior • Metrowerks	BC
c-tree Plus • FairCom Corporation	7
Dave • Thursby	63
Development Tools • DevDepot	28-29
Dr. Bott	51
edition.net	39
eSellerate • Mindvision	47
Equipment - KVM, FireWire, USB • IOGEAR	87
FastTrack Schedule • AEC	25
Fetch • Fetch Softworks	8
Funnel Web 3 • Quest	17
Installer Maker • Aladdin Systems	9
iTunes • Dr. Bott	65
Job Placement • Scientific Placement	19
Lego Mindstorms • Pitsco-Lego Dacta	37
LexTek	39
Linux • SuSE	IBC
Localization • Geotopia	76
MacHasp USB • Aladdin Knowledge Systems	55
Mac OS X Books • O'Reilly	89
Mac Software Development & Testing • Full Spectrum	38
MacAddict	68
Macworld Conference & Expo • IDG World Expos	56-57
Monster Products • ASC Tech	14
NetOctopus • Netopia	21
OO File • AD Software	19
PrimeBase	53
Rackmount solutions and accessories • Marathon Computer	73
RAD Studio • OpenBase International	13
Radio Show • Mac OS Radio	35
Rapid MQ • Jiiva, Inc.	15
REALbasic • REAL Software	49
Resorcerer • Mathemaesthetics	45
Rewind • PowerOn Software	71
SkyLINE 11MB • Farallon	77
Sound Studio • Felt Tip Software	91
Spotlight • Onyx Technologies	48
Stone Table • Stone Table Publishing	27
SuperReport Pro 2.5 • Automated Solutions Group	5
Tera Media • Tera Global Communications	IFC
Valentina • Paradigma	31
VISE • Mindvision	33
VOODOO Server • UNI SOFTWARE PLUS	61
WEB 2001 • CMP Media	74-75
WIBU-KEY Software Protection • WIBU SYSTEMS AG	69

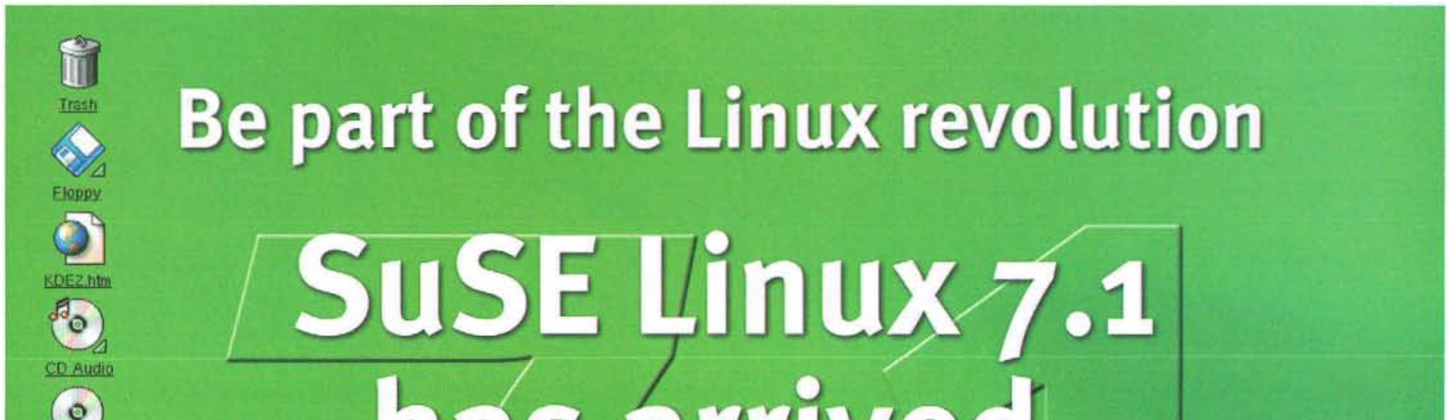
Mac OS X Porting and Development Showcase

Atimi	84
Art & Logic	83
Omni Group, The	82
Parallel	80
ProSoft Engineering	78-79
Vanteon	81

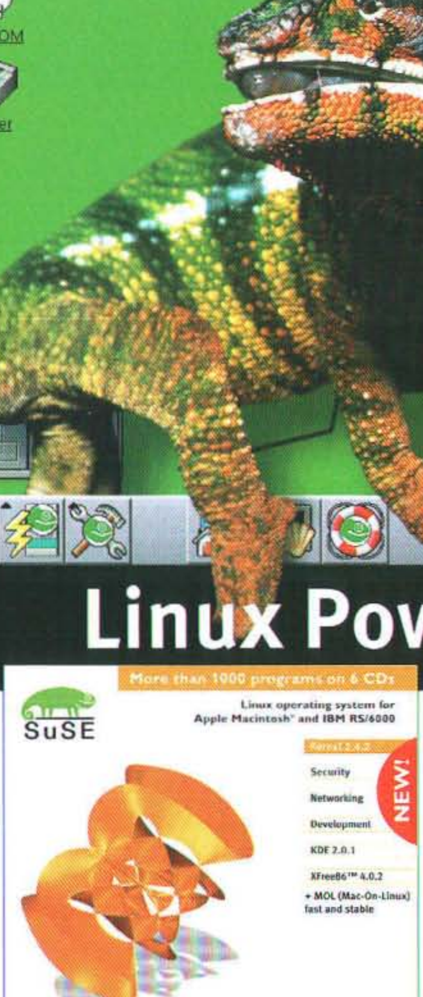
Mac OS X Porting and Development Showcase

Consulting • Omni Group, The	82
Mac OS X/9.x/8.x Development • Vanteon	81
Mac OS X Migration • Parallel	80
Mac OS X Porting • ProSoft Engineering	78-79
Software Development • Atimi	84
Software Engineering • Art & Logic	83

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.



Linux Power for your PowerPC


More than 1000 programs on 6 CDs
SuSE
Linux operating system for Apple Macintosh® and IBM RS/6000
NEW!
Security
Networking
Development
KDE 2.0.1
XFree86™ 4.0.2
+ MOL (Mac-On-Linux) fast and stable
SuSE LINUX 7.1
POWERPC EDITION
60 days of free installation support

From graphics to software development and even server implementations, SuSE offers **enterprise-level performance** without the price tag.

SuSE Linux 7.1 PowerPC EDITION combines remarkable efficiency with unparalleled flexibility to provide you with one of the most valuable operating system packages on the market today.

Get comfortable with the **MOL** (Mac-On-Linux) emulator and enjoy your network-capable MacOS under Linux in window or fullscreen mode.

The user-friendly quick-start-menu within the latest **KDE 2.0.1** desktop environment will easily activate your e-mail program, your organiser tool or your media-player; everything is included with the "drag & drop" functionality.

Our **expert team** offers you 60 days of individual installation support via phone, fax or email.

SuSE Inc.
580 Second Street
Oakland, CA 94607

info@suse.com
Phone: (888) UR-LINUX
(510) 628-3380
Fax: (510) 628-3381

Place your order today! www.suse.com

SuSE

Keeping Mac dreams alive since 1993.

Two great operating systems, one CodeWarrior.

Mac developers depended on CodeWarrior to make the platform architecture shift from 68K to PowerPC processors. Now CodeWarrior does it again, speeding your transition to the next new operating system. CodeWarrior for Mac OS, Version 6.0 supports development for both OS X and Classic Mac operating systems from a single, powerful, award-winning Integrated Development Environment. Discover how CodeWarrior for Mac OS, Version 6.0 can help you realize your Mac development dreams. Visit **www.metrowerks.com/go/mac**.

CodeWarrior®

metrowerks®
Software Starts Here